USENIX

PROCEEDINGS

# of the

Second Computer Graphics Workshop

Monterey 1985

USENIX Association

Second Computer Graphics Workshop

Monterey 1985

PROCEEDINGS

December 12-13, 1985

Monterey, California, USA

For additional copies of these proceedings, or copies of
the Proceedings of the First Computer Graphics Workshop, write:

USENIX Association
P.O. Box 7
El Cerrito, CA 94530 USA

The price per copy is $3.00 plus $7.00 for overseas airmail.

ACKNOWLEDGMENTS

Sponsored by:	USENIX Association P.O. Box 7 El Cerrito, CA 94530	
Program Chair:	Reidar Bornholdt	Columbia University
Program Committee:	Tom Duff	AT&T Bell Laboratories
	Lou Katz	Networked Picture Systems
	Peter S. Langston	Bell Communications Research
Workshop held at:	Doubletree Hotel Monterey, California	
USENIX Meeting Planner:	Judith F. DesHarnais	
Proceedings Production:	Peter H. Salus	USENIX Executive Director
	Tom Strong	Strong Consulting

WORKSHOP PROGRAM AND TABLE OF CONTENTS

Attendee's Session

A Data-flow Environment for Interactive Graphics

Paul E. Haeberli

Silicon Graphics Inc.
Mountain View, California 94043

ABSTRACT

Operating system extensions are described that allow programs to be combined in a flexible and powerful way. A *data-flow manager* is developed to control the flow of data between processes. This system provides a method for the dynamic interconnection of processes running under a window manager. The interconnection of processes may be edited interactively, and no limitations are placed on the structure of process interconnection. This environment encourages the creation of simple, modular tools that work well together.

1. Summary

The primary motivation for this work was the growing complexity of interactive programs running on the IRIS workstation. The IRIS workstation consists of a 68020 processor with memory, disk, Ethernet, and custom graphics hardware.

Many programs require that an object be viewed with different rotations and scale. Significant work was needed to parse mouse movements, and generate proper views of the object. At the same time it was desired that interaction be expanded to allow a sequence of views to be saved or replayed, and that additional controls over illumination direction also be provided. The thought of even more complex interaction handlers was unattractive. This motivated work to develop a more synergetic environment where specific tools could be used to change and record views of objects, and their illumination.

The following material was used in a presentation describing this environment.

Synergy

Synergy means the behavior of whole systems that is unpredicted by the behavior of their parts taken separately.

R. Buckminster Fuller

It's a New World!

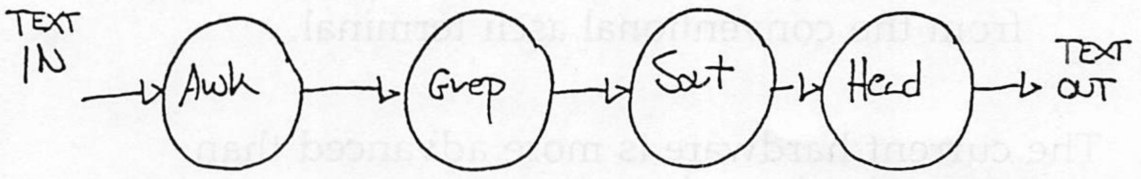
The current generation of graphics systems differ greatly from the conventional ascii terminal.

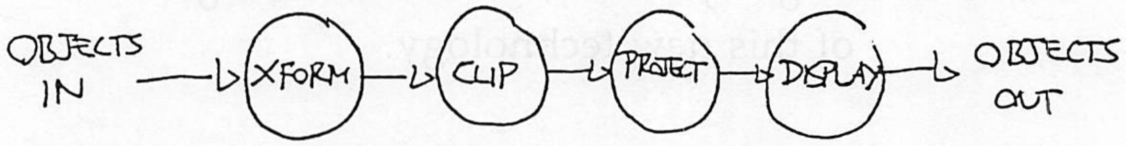
The current hardware is more advanced than the software running on it.

We must design systems that take advantage of this new technology.

Applications of pipes

Great for doing processing that maps into a pipeline



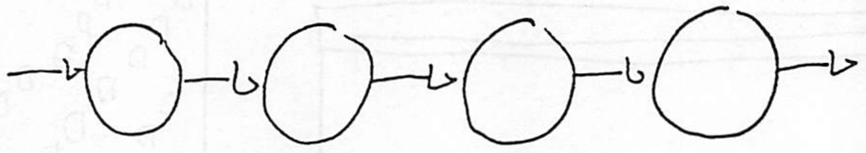


Model breaks down if we want to modify the function of one component of a pipe, or alter the flow of data.

Problems with Unix Pipes

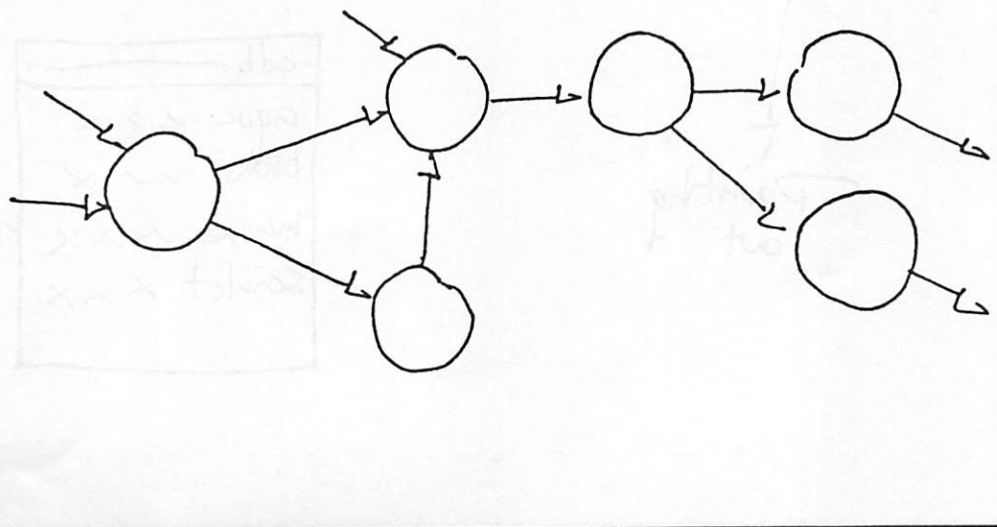
Unix Pipes

One input and one output
Fan-in is 1 and fan-out is 1
Interconnection is static
No interactive control of pipe components

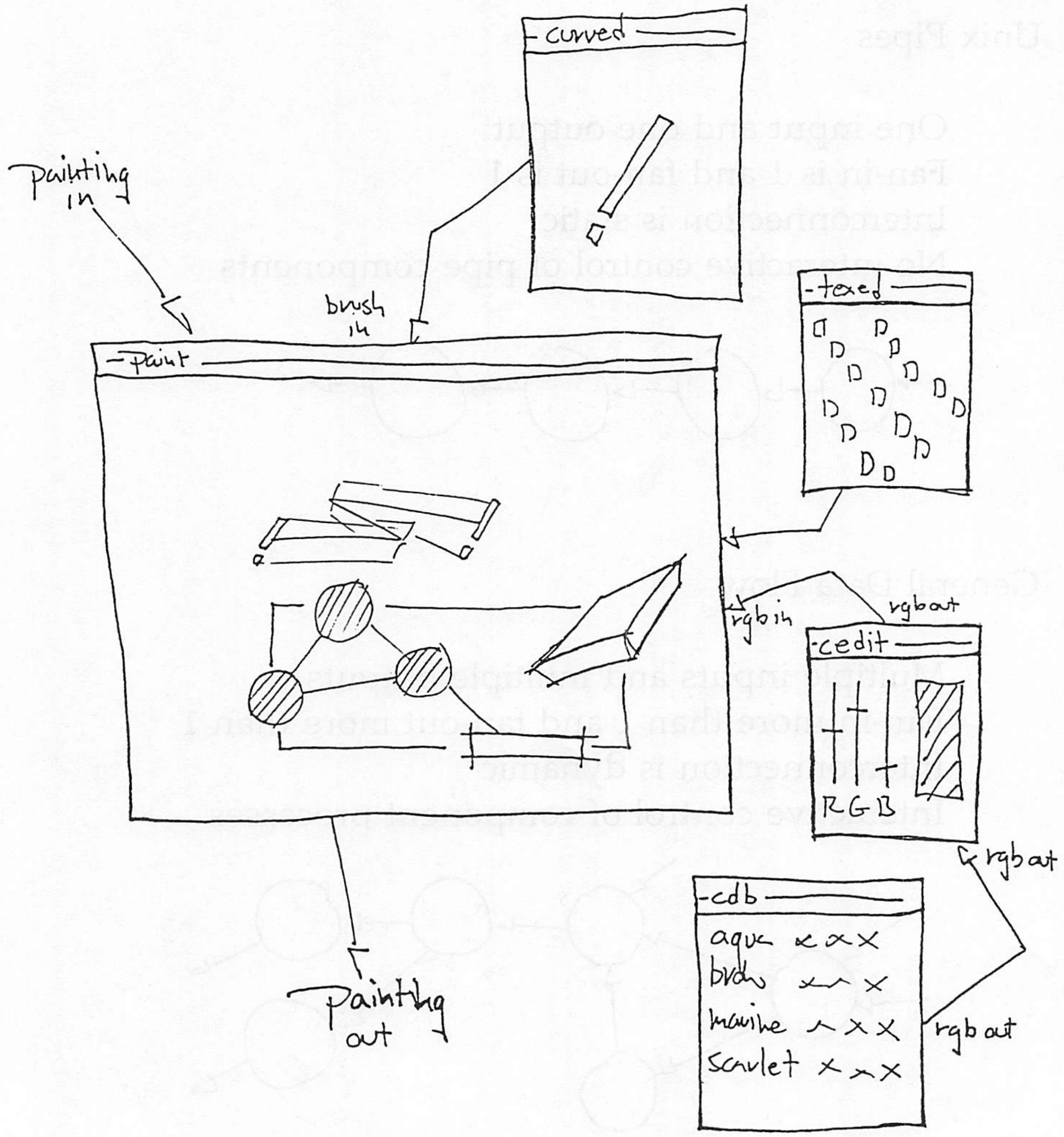


General Data Flow

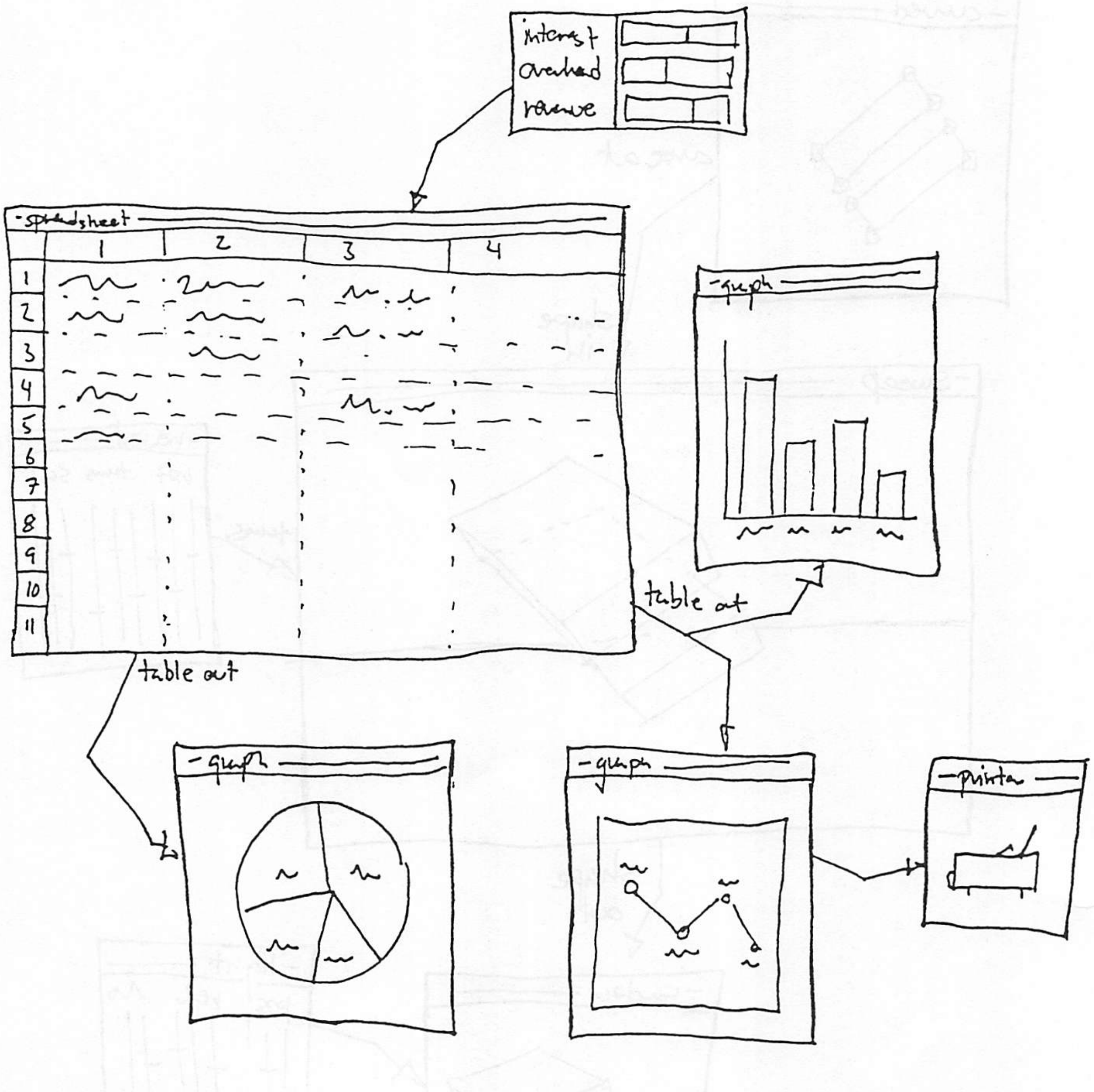
Multiple inputs and multiple outputs
Fan-in more than 1 and fan-out more than 1
Interconnection is dynamic
Interactive control of component processes



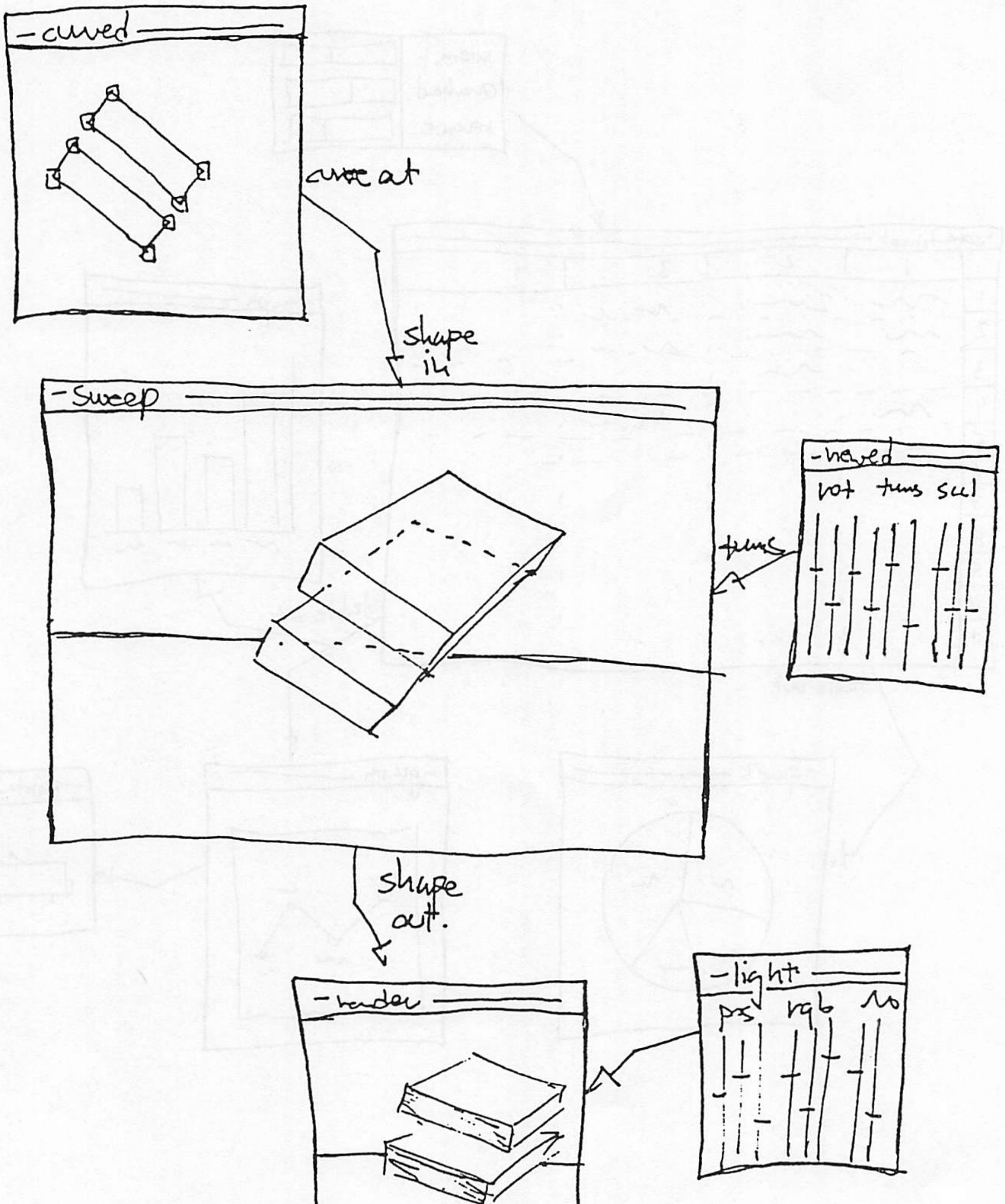
A set of processes in a general data flow enironment



A Spreadsheet



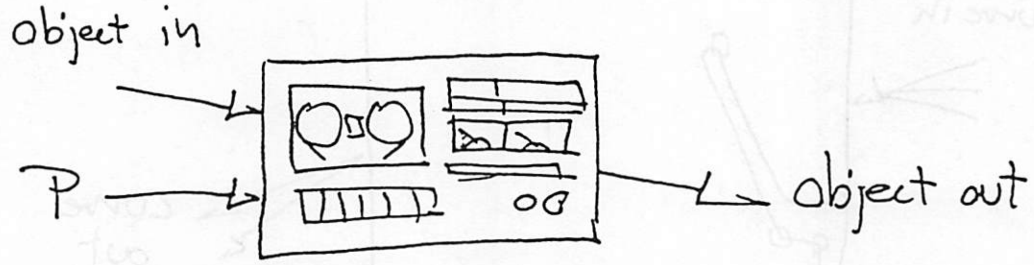
A Graphic Synthesizer



NAME

tape - record and playback a series of objects

VIEW



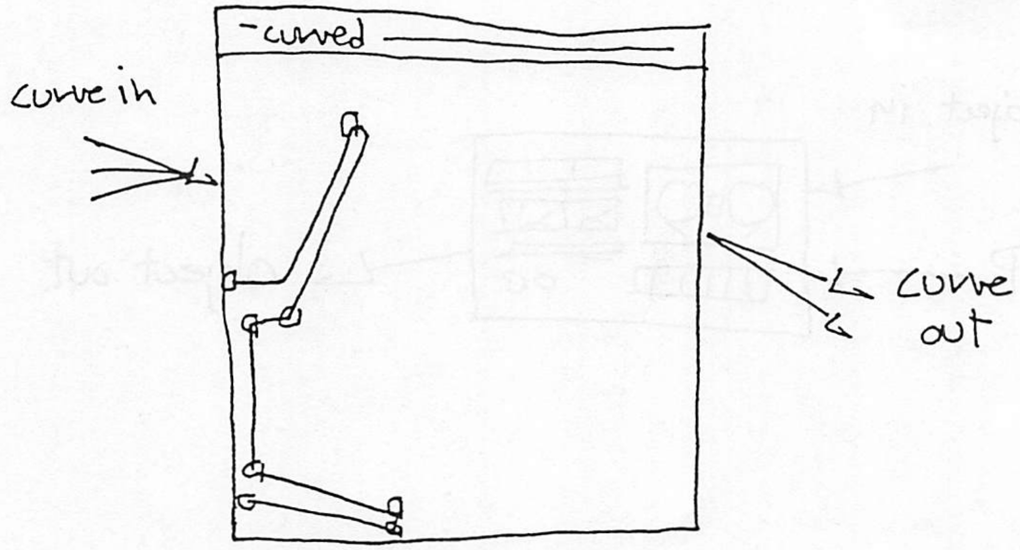
DESCRIPTION

Tape allows a series of objects to be recorded for future play back. This tool can be used to record a series of views, or a series of geometric objects. Tape has two input ports: one for the input object stream, and one for a parameter p which may be used to seek on the tape. The output port is used to transmit the contents of the tape when playing, or looping.

NAME

curved - edit a 2 dimensional set of lines

VIEW



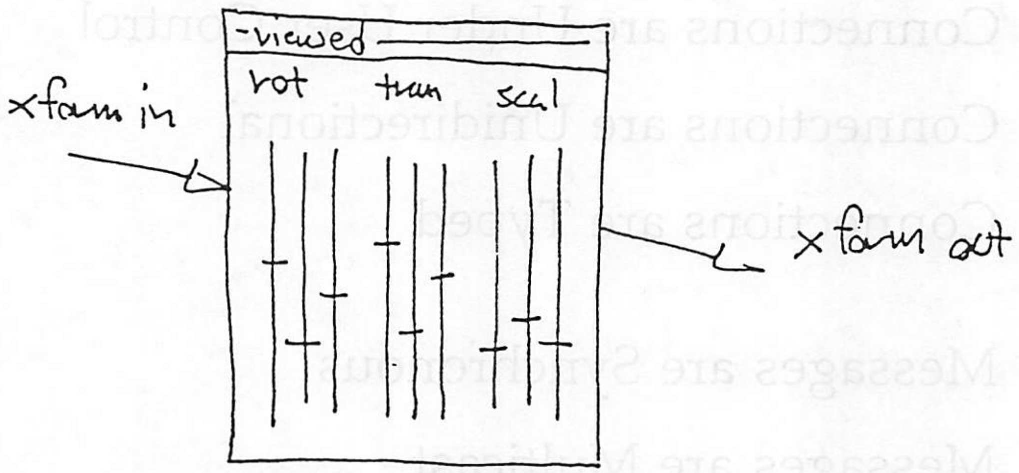
DESCRIPTION

Curved allows planar points, lines and polygons to be edited interactively. It has one input port and one output port for the curve being edited. The curve is represented in ascii model format

NAME

viewed - edit a set of transformations

VIEW



DESCRIPTION

Viewed consists of a set of sliders that may be used to control rotation, translation, and scale of geometric objects. It has one output port for the current set of transformation commands. The output is represented in ascii model format.

Features of this Environment

Allows General Connection Of Processes

Connections are Under User Control

Connections are Unidirectional

Connections are Typed

Messages are Synchronous

Messages are Multicast

Messages are Variable Sized

A Low Cost Graphics Workstation*

Spencer W. Thomas
Computer Science Dept.
University of Utah
Salt Lake City, Utah 84112
USA

Abstract

The Apple Macintosh is used as the basis for a low-cost, multi-function graphics workstation. It provides multiple graphics and text editing windows, with local 3-D display lists and transformations. The workstation is used as a front-end interface to the Alpha_1 geometric modeling system, but can be easily driven by other graphics applications.

* This work was supported in part by the National Science Foundation (DCR-8341796 and MCS-8121750), the Defense Advanced Research Projects Agency (DAAK11-84-K-0017), and the Office of Naval Research (N00014-82-K-0351). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

1. Introduction

The Alpha_1 project at the University of Utah is engaged in developing a computer-aided solid modeling system [4, 3]. The core of the system is a lisp based geometric editor called ShapeEdit (Figure 1). ShapeEdit supports a number of graphics displays, ranging from Evans and Sutherland PS300 and Silicon Graphics Iris to Apollo bitmap displays. However all of these displays are relatively expensive, and are generally only found in the context of a large installation. It was felt that a low-cost graphics display that would be portable, and could be used "at home", would be a useful addition to the available repertoire. If the display were to be the sole interface between the user and ShapeEdit, it should also support a local text-editing functionality and a file-transfer capability.

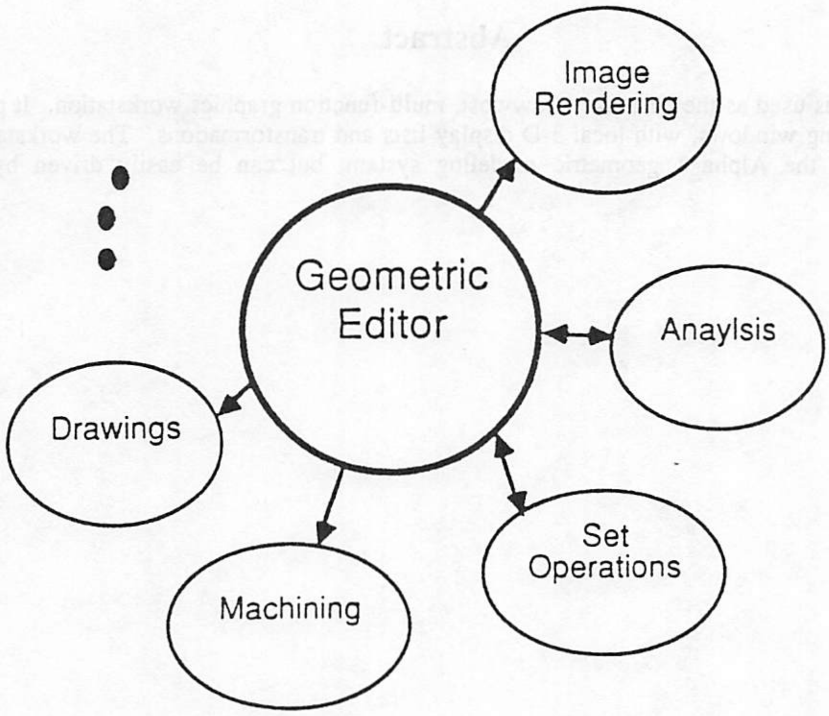
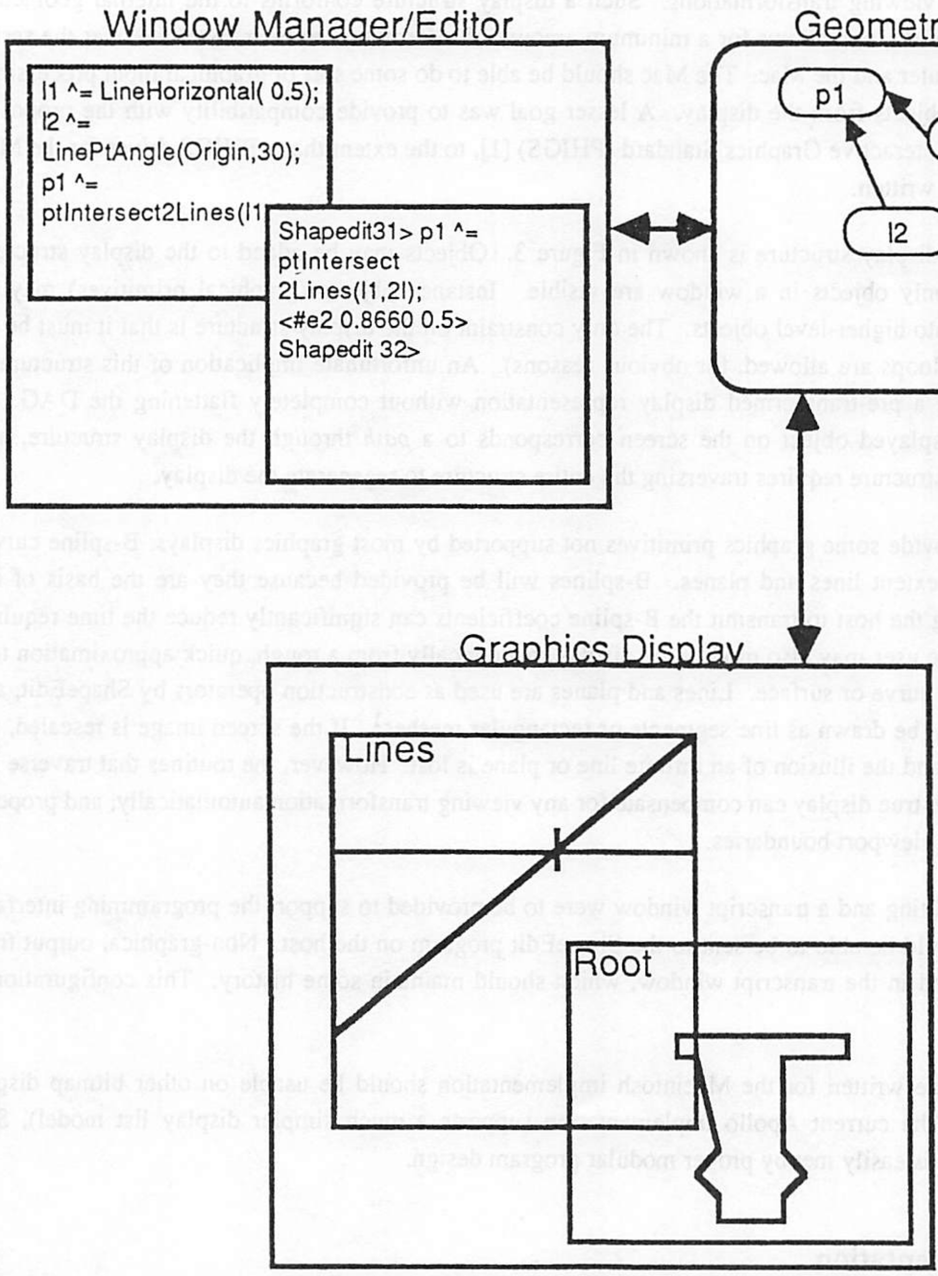


Figure 1. The Alpha_1 Solid Modeling System

The availability of the Apple Macintosh, a low-cost computer with reasonably high-resolution bitmap graphics, prompted an investigation into its suitability as such a graphics workstation. The Macintosh multiwindow user interface was similar to the most common mode of ShapeEdit operation -- running it as a subprocess of a window based text editor, with multiple text editing windows, a transcript window with history, and multiple graphics windows. Figure 2 shows the current model of interaction with ShapeEdit. The part being designed is described by writing statements in a programming language. While is a powerful description technique, it is not very "user-friendly". Work is currently underway to provide a graphical interface in addition to the current programming interface. Thus, the workstation should provide graphical input facilities, including simulation of locator, valuator, button or menu, and pick inputs. Preliminary results indicate that the Apple Macintosh can meet the goals of providing a low cost, medium performance 3-D graphics workstation.



```
l1 ^= LineHorizontal( 0.5);
l2 ^=
LinePtAngle(Origin,30);
p1 ^=
ptIntersect2Lines(l1
```

```
Shapedit31> p1 ^=
ptIntersect
2Lines(l1,2l);
<#e2 0.8660 0.5>
Shapedit 32>
```

Figure 2. Current interaction model

2. Design Goals

The major design goal for the project was to provide a graphics workstation with a 3-D hierarchical, editable display structure and local viewing transformations. Such a display structure conforms to the internal geometric model maintained by ShapeEdit, and allows for a minimum amount of information to be transmitted over the serial link between the host computer and the Mac. The Mac should be able to do some sort of graphical input processing, including the ability pick objects from the display. A lesser goal was to provide compatibility with the proposed Programmers Hierarchical Interactive Graphics Standard (PHIGS) [1], to the extent that a PHIGS driver for the Mac workstation could be easily written.

An example of the 3-D display structure is shown in Figure 3. Objects may be added to the display structure without being displayed; only objects in a window are visible. Instance objects (graphical primitives) may be transformed and grouped into higher-level objects. The only constraint on the display structure is that it must be an acyclic directed graph (no loops are allowed, for obvious reasons). An unfortunate implication of this structure is that it is difficult to cache a pre-transformed display representation without completely flattening the DAG. A particular instance of a displayed object on the screen corresponds to a *path* through the display structure, any modification to part of the structure requires traversing the entire structure to regenerate the display.

The workstation will provide some graphics primitives not supported by most graphics displays: B-spline curves and surfaces, and infinite extent lines and planes. B-splines will be provided because they are the basis of the Alpha_1 system. Allowing the host to transmit the B-spline coefficients can significantly reduce the time required to download an image. The user may also modify the display mode locally from a rough, quick approximation to a fine, but slow image of the curve or surface. Lines and planes are used as construction operators by ShapeEdit, and on all current displays must be drawn as line segments or rectangular meshes[1]. If the screen image is rescaled, the endpoints become visible, and the illusion of an infinite line or plane is lost. However, the routines that traverse the display list and compute the true display can compensate for any viewing transformation automatically, and properly clip the line or plane to the viewport boundaries.

In addition, local text editing and a transcript window were to be provided to support the programming interface. Data in a text window should be able to be sent to the ShapeEdit program on the host. Non-graphical output from ShapeEdit will be displayed in the transcript window, which should maintain some history. This configuration is shown in Figure 4.

Finally, most of the code written for the Macintosh implementation should be usable on other bitmap display devices, such as Apollo (the current Apollo implementation supports a much simpler display list model), Sun, VaxStation, etc. This goal is easily met by proper modular program design.

3. Macintosh Implementation

A preliminary version has been written, incorporating only a single text editing window and a single graphics window, and rudimentary communications facilities, but implementing the full 3-D display list and viewing operations. The graphics is fast enough that redisplay of a complex object, such as that shown in Figure 5, takes a few seconds. Simple objects with a few tens of vectors can be redrawn in "real time". This is acceptable performance, and is in line with prior expectations. The code is all written in C, and Macintosh-specific functions have been strictly isolated from machine-independent functions.

3-D line drawings on a bitmap display with no hidden lines removed can be difficult to interpret. One feature of a

[1]Anyone who has a better way to display a plane, please tell me.

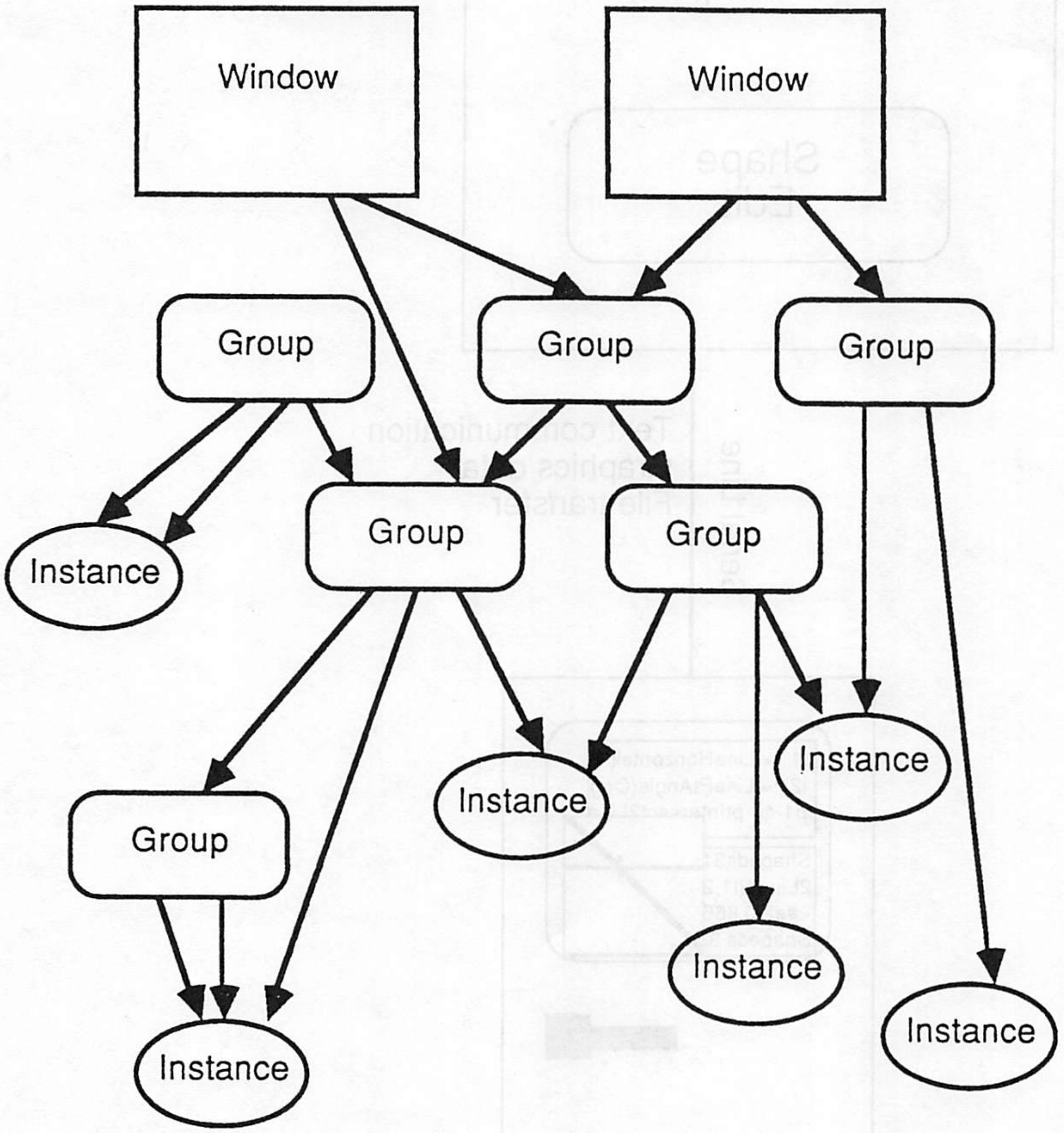


Figure 3. 3-D display structure

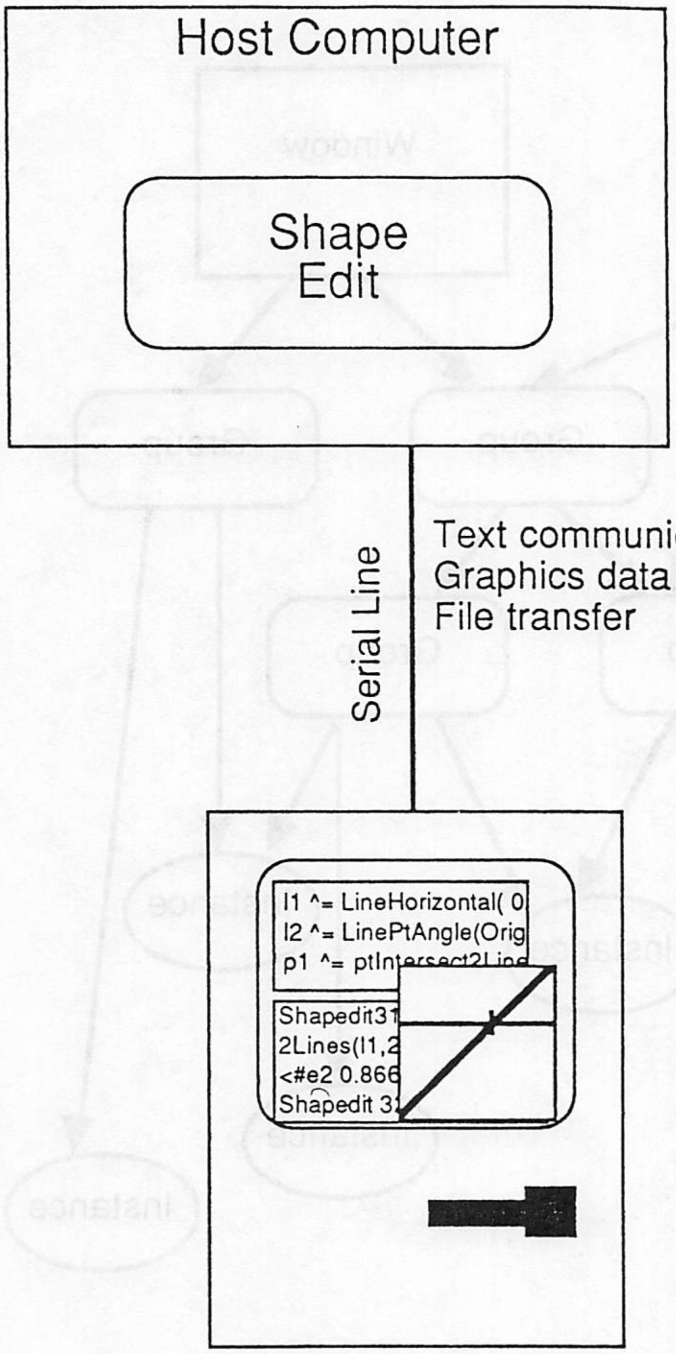


Figure 4. Macintosh workstation configuration

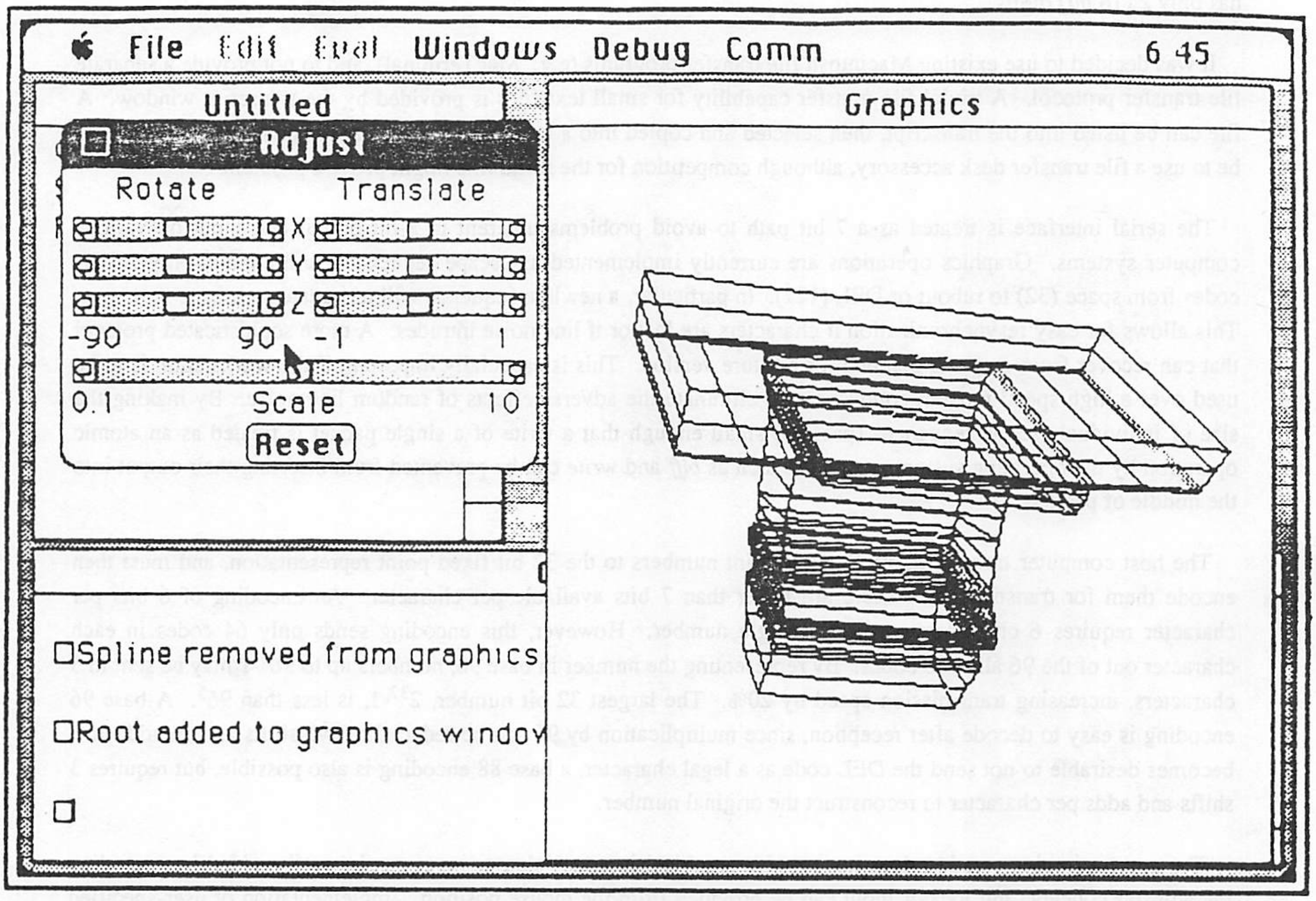

File Edit Eval Windows Debug Comm 6 45
Untitled
Graphics
Adjust
Rotate
Translate
X
Y
Z
-90 90 -1 1
0.1 Scale 10
Reset
Spline removed from graphics
Root added to graphics window

Figure 5. Sample screen display

graphics display that can aid perception of 3-D shape, and that can be used to disambiguate back and front, is depth cueing. Experimentation has shown that a single "bit" of depth information greatly enhances perception of depth on a bitmap display. This can be easily provided by using a dotted line style for all line segments behind the Z=0 plane of the display.

Local viewing transformations are provided by an "adjuster" window. At the moment, all transformations are implemented by incremental sliders, as shown in Figure 5. Each slider provides a transformation in the screen coordinate system. That is, all rotations occur about the X, Y, and Z axes of the window (the origin is always centered in the window), and translations similarly are always parallel to the window coordinate axes. After selecting a slider and moving its "thumb", when the user releases the mouse button, the view in the graphics window is immediately updated, and the thumb returns to the center of the slider. The scaling slider has a logarithmic scale, so that two clicks at a given position on the slider have exactly the same effect as a single click twice as far from the center. At some point, the slider model will probably be replaced by more intuitive controls, especially for rotation. The adjuster controls may also be integrated into each graphics window.

All arithmetic is done using fixed point operations to avoid the slow speed of floating point emulation. The Macintosh Toolbox provides a set of operations on 32 bit fixed point numbers, omitting only division. The multiple precision division algorithm from *Knuth*, Volume 2 [2] was used to fill this lack. The algorithm was modified slightly to produce results with a properly placed binary point, and was optimized for the case where the dividend has only 2 (16 bit) digits.

It was decided to use existing Macintosh file transfer programs (e.g. MacTerminal), and to not provide a separate file transfer protocol. A trivial file transfer capability for small text files is provided by the transcript window. A file can be listed into the transcript, then selected and copied into a text edit window. A possible alternative would be to use a file transfer desk accessory, although competition for the serial line might prove a problem.

The serial interface is treated as a 7 bit path to avoid problems inherent in 8-bit output and input on various computer systems. Graphics operations are currently implemented as escape sequences, and contain only ASCII codes from space (32) to rubout or DEL (127). In particular, a newline sequence will terminate an escape sequence. This allows for easy resynchronization if characters are lost or if line noise intrudes. A more sophisticated protocol that can recover from errors is planned for a future version. This is especially important if the workstation is to be used over a high speed modem connection, to eliminate the adverse effects of random line noise. By making the size of individual escape sequences (packets) small enough that a write of a single packet is treated as an atomic operation by the operating system, programs such as *biff* and *write* can be prevented from inserting their output into the middle of packets.

The host computer transforms its floating point numbers to the 32 bit fixed point representation, and must then encode them for transmission. There are fewer than 7 bits available per character. An encoding of 6 bits per character requires 6 characters to send a single number. However, this encoding sends only 64 codes in each character out of the 96 allowed codes. By representing the number in base 96, numbers up to 96^5-1 may be sent in 5 characters, increasing transmission speed by 20%. The largest 32 bit number, 2^{32}-1, is less than 96^5. A base 96 encoding is easy to decode after reception, since multiplication by 96 can be reduced to two shifts and an add. If it becomes desirable to not send the DEL code as a legal character, a base 88 encoding is also possible, but requires 3 shifts and adds per character to reconstruct the original number.

The current implementation does not support graphical input. Valuator input can be easily added by extending the adjuster concept, and locator input can be provided from the mouse position. Implementation of user-specified menus is planned, up to the number of menus that will fit on the Mac's menu bar. Selecting a menu item from one of these menus will send a string to the host signaling the selection. Finally, object picking is also planned, and will be implemented by a "hit detection" method. As the display list is traversed, each primitive is checked to see if it crosses a small window centered at the mouse position. If it does, it is added to the "hit list". Besides being able to set detectability at a group level, it will be possible to select hit detection on a particular type of primitive (e.g., point, polyline, curve, etc.). Objects hit are identified by the path from the display list root to the instance containing the object.

4. Conclusions

A preliminary version of a Macintosh graphics workstation has been implemented, and shows satisfactory performance. The total code size for this version is about 50k bytes, leaving room for large display lists in a "Fat Mac", and a reasonable amount of space in a Mac with only 128K of memory. The low cost of the Macintosh compared to other 3-D graphics displays renders this an attractive option for portable or home use. The initial version is missing some planned features, including graphics input and multiple windows. These will be included in future versions. The Mac has been added to the "stable" of 3-D graphics workstations.

Note: this program will be made available. Contact Spencer Thomas for details.

Email: thomas@utah-cs.ARPA, {ihnp4,decvax}!utah-cs!thomas
Phone: (801)484-8944
USmail: Spencer W. Thomas
Computer Science Dept.
University of Utah
Salt Lake City, Utah 84112

References

[1] ANSI X3H3.
American National Standard for the Functional Specification of the Programmer's Hierarchical Interactive Graphics Standard (PHIGS).
American National Standards Institute, 1985.
Draft revised 2/18/85.

[2] Knuth, D. E.
The Art of Computer Programming. Volume 2: *Seminumerical Algorithms.*
Addison-Wesley, 1981.
Second Edition.

[3] Riesenfeld, Richard F.
A View of Spline-Based Solid Modelling.
In *Autofact 5 Conference Proceedings.* Computer and Automated Systems Association of SME, SME, 1983.

[4] Thomas, Spencer W.
The Alpha_1 Computer-Aided Geometric Design System in the UNIX Environment.
;login: 10(4):54-64, October/November, 1985.
Presented at the 1984 USENIX Graphics Workshop.

MacMix:Mixing Music with a Mouse

Adrian Freed
22 Sirard Lane, San Rafael, CA 94901[1]

ABSTRACT

MacMix is a user interface running on an Apple Macintosh computer, which communicates to processes on a UNIX machine that do the work of mixing and playing files of sound. In many ways the architecture of MacMix resembles that of airline reservation systems or bank networks, where a local terminal initiates transactions on a remote database. MacMix differs from these in that the remote database is physically close, and that the graphics user interface requires rapid response in bursts from the network. A video tape is available which shows MacMix in action. This medium is much better than the written word for communicating the details of a graphical user interface. This paper will focus on the unanticipated design and implementation problems of the project, most of which are related to networking.

Introduction

MacMix was developed at IRCAM, a contemporary music institute in Paris. Musicians are interested in using computers to synthesise, analyse and modify sounds. The process of mixing or blending separate sounds together is fundamental and very common in musical production of all sorts.

Most of the difficulties we face with using computers to manipulate sound stem from a simple fact: we do not have a very compact digital representation of sound. We need roughly 1Mbyte of space to store 10 seconds of hi-fi quality sound. IRCAM has several Gigabytes of on-line disk storage and a large library of sound on disk packs and magnetic tape.

Most installations with this amount of data to store and process have found that a few, very large, centralised systems are more cost effective than a network of distributed workstations. This may change in the next few years, but musicians seek tools today and they are seduced by convenience and efficacity of mouse-graphic based workstations. MacMix attempts to give a musician the control they seek over sound stored on a centralised system shared by other musicians. The architecture chosen was to use a popular mouse-graphics computer for the user interface and have it talk over a network to IRCAM's large sound file host computer.

There are many such small computers to choose from. We chose the Macintosh because it is physically small, reliable, fairly cheap and has no noisy fan. Fan noise is inconvenient in recording studios. It would have perhaps been easier to use a UNIX workstation, since sounds at IRCAM are stored on large UNIX systems and off-the-shelf solutions exist for UNIX to UNIX communication problems. However, we will see that the network requirements of MacMix are not very cost-effectively served by such solutions.

The User is in Control

Users should feel that they are in control of MacMix and that services requested from the host result from their gestures. This requirement led to the single most important feature of the communication protocol between Macintosh and host: the Macintosh is always the master, the host is a slave.

The Macintosh sends requests for service to the host and waits for one of three kinds of responses: request serviced, request not serviced, and no response at all. The slave host computer only sends messages to the Macintosh in response to a request. It cannot request service from the Macintosh.

1. This work completed at IRCAM. Electronic mail correspondence to: seismo!ircam!adrian.

* UNIX is a trademark of ATT Bell Laboratories.

* Macintosh is a trademark of McIntosh Laboratories, used by permission.

Stateless Server

The second requirement of the protocol is derived from the observation that the host is not to be trusted. The host should not be trusted because it is less reliable than the Macintosh and it is not under the control of the user. It need not even be in the same building or country as the Macintosh.

If the host can not be trusted, MacMix has to behave coherently when requests are not serviced and when the host fails to respond. It should also not be too sensitive to errors resulting from a host claiming it performed a service when it actually did not. These observations lead to the use of a stateless server on the host. If the host cannot be trusted, we should not assume that it can remember anything of early requests. This means that each request from the Macintosh has to include all the information needed for it to be completed by the host.

An example of state stored in the Macintosh which could have been stored on the host is the current directory. All file names sent from the Macintosh are also sent with a directory path. Directories are managed by the Macintosh even though their contents exist on the host.

Performance

In the long term MacMix uses very little of the available network bandwidth. This is because much of what the user is doing is provided for on the Macintosh. Every few seconds MacMix will issue a request for the contents of a directory or a few hundred samples in a sound file. The performance problem is that the user will be twiddling their thumbs as soon as the request is sent and will not stop twiddling until something changes on the screen in function of the hosts reply to the request. In other words latency is far more important than bandwidth in this application.

This can be contrasted with networks used for moving files and mail from system to system. In these applications delays of minutes to hours may be acceptable, but there may be a large amount of data involved. MacMix requests are rarely longer than a 100 characters and replies rarely more than a few thousand characters.

Implementation

MacMix has been successfully used with a very simple communication protocol running on RS232 communication lines at 19,200 Baud. The protocol is simple because we made the seemingly rash assumption that our communication lines were error free. The protocol has no error-correction. It has poor error detection facilities, but error recovery is very good since the Macintosh is so suspicious of the host.

The assumption of error free lines is a good one, if the lines are installed correctly and operate over short distances. We have only observed one kind of error resulting from use of these lines: total failure. These errors occur for silly reasons: the cable is pulled from back of the Macintosh, someone accidentally unpatches the cable at the host, or rats eat through the cable. Whatever the cause, no amount of clever error-correcting will repair the cable.

What if we wanted to run MacMix remotely? In France it is cheaper to use a national packet-switching network to access IRCAM than to use modems. This network provides error-free transmission. In the USA it would be necessary to use error-correcting modems. We do not feel that this is a serious limitation as conventional 1200 baud modems are too slow for this application.

Improvements

Several tools would have made implementation easier and more powerful. It is surprising that UNIX is not already supplied with a protocol similar to the one we devised. Many sites use UNIX to down or up load data to other systems.

UNIX System V comes with the xt driver, which is used to communicate with software in the DMD terminal. Their protocol appears to be useful in applications like MacMix, but has not been published. Its main feature is that it multiplexes several channels on a single RS232 communication line. If this was available MacMix could control several host computers simultaneously, or users could interact with several UNIX processes at once on a single Macintosh.

We use XON/XOFF flow control in both directions to avoid overflowing input buffers in the Macintosh and UNIX host. The UNIX host we are using, like many available, finds it buffers overflowing far more often than necessary. These systems cope with data rates from disks at least 10 times than those from serial ports. That latest generation of UNIX machines do not appear to have this problem.

The MacMix server process is structured much like a UNIX shell. It parses a request. If the request makes sense it checks to see if another process is required to perform the request. If there is it handles communication with that process. If not it fulfills the request directly. Why not use the shell? Why did we create our own server process? The problem with the shell is that there is no easy way a program can parse its output and the output of the programs it executes. This output was designed for human consumption. There is a also a performance problem, as the shell spawns new processes for each user request. Our implementation spawns a single process for each service required and communicates directly to these processes. There is no overhead for process creation and death and these processes are able to cache open files.

Conclusion

We believe that many interesting applications will be structured the way MacMix is. We have discussed some of the difficulties raised by such a structure. The rewards of overcoming them are hopefully demonstrated in the video tape of MacMix in action.

Acknowledgement

The UNIX host software for MacMix was implemented by Robert Gross.

An

Adrian Freed

IRCAM

Coproduction

Adrian Freed
22 Sirard Lane
San Rafael, CA94901

seismo!ircam!adrian

Robert Gross
IRCAM
31, rue Saint Merri
75004 Paris, France

seismo!ircam!robert

The Problem

Computerise

control,
storage,

and

manipulation

of sound in recording studios

Why?

lower costs
increase quality
reduce production time
reduce physical handling
interesting challenge

NEW AVENUES OF ARTISTIC EXPRESSION

<u>Elaboration:</u>

<u>Storage requirements I</u>

The 4-minute rock song

4 minutes
60 seconds/minute
24 tracks
100k bytes/second/track

=> 676 Megabytes
+ ECC and overhead
=> 1 Gigabyte

Elaboration:

Storage requirements II

Ircam: contemporary music

40 minutes/track
12 tracks

8 studios
=> 5 Gigabytes

or

20 musicians
=>80 Gigabytes

Elaboration:

Input/Output performance I

Requirements:
- rock:
  - 32 tracks is common
  - final mix: stereo
- contemporary:
  - <16
  - final mix:quad

4.2BSD on Vax 750 or 780:
- 4 channels
- at 32Ksamples/second
- (Industry standards
- are 48K and 44.1K)

Elaboration:

Input/Output performance II

Graphics workstations:

  Sun, Mac, Amiga etc.

  1 million pixels to view
  gigabytes database

Networks:

  Ethernet

  2 channels at 48Ksamples/s
  ?Anyone done this?

A SOLUTION: MacMix

The video tape:

for training musicians

poor quality standards conversion

notice the performance (fudged)

Hardware Architecture

Centralised storage and processing of sound
->economies of scale

Distributed control and viewing station, e.g. Macintosh
-> small
-> cheap
-> plentiful
-> reliable
-> known
-> no fan (yet)

RS232 communication link
-> cheap
-> error free over short distances

Software Architecture

MacMix Macintosh application makes requests to host server

MacMix provides terminal emulation for host operations not supported by server

Server process sub-contracts some requests to mix(1) and play(1) over a socket(2) pair

Server uses caches to improve performance, e.g. envelope file

PC/UNIX networking

Stateless host server.

Automated login and validation.

Latency is more important than burst or long term throughput.

A sh(1) for programs:
process startup/exit,
standard output language.

Simple public domain protocol for multiplexing, e.g. xt(4).

Flow control on UNIX input side.

How much network overhead is justified?

Conclusion

If you are looking for a source of difficult problems in:

data bases
graphics
signal processing
workstation architecture
user interfaces
and
networks

digital audio is a place to look. It is also a lot of fun.

BERKELEY UNIGRAFIX
A MODULAR RENDERING AND MODELING SYSTEM

Carlo H. Séquin

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

ABSTRACT

Berkeley *UNIGRAFIX* provides simple graphics and modeling capabilities for 3-dimensional polyhedral objects within the UNIX operating system. It renders mechanical parts or geometrical manifolds in the style of engineering drawings, with visible edges displayed as solid lines and faces shaded to enhance understandability of the drawing, rather than to make the objects look "natural". Output is primarily aimed at high-resolution black-and-white dot-raster plotters but can also be previewed on various terminals. Because of the potentially large number of pixels (50 million pixels for a 3 foot by 3 foot Versatec plot), the hidden face and line algorithms are designed for efficiency, exploiting scan-line coherence as well as object coherence.

The Berkeley *UNIGRAFIX* system also comprises a set of generator programs that assist in the creation of parameterized object descriptions such as gear wheels or architectural elements (staircases, houses ...). There are programs that modify simple objects by truncating them, by tessellating their faces, or by cutting holes into them; others place pyramids on all faces or replace the edges of the object with thin prismatic solids. This set of *UNIGRAFIX* routines and the associated data structures form the basis of a solid modeling environment.

All these programs are loosely tied together by the UNIX 4.2 BSD operating system and by a simple descriptive ascii format for the specification of the geometrical objects. This *UNIGRAFIX* object description format defines geometry by specifying all vertex coordinates, and then linking edges and faces to these vertices by reference. It also contains the necessary macro definition facilities to permit the description of complicated scenes in a hierarchical manner.

1. INTRODUCTION

With the rapidly growing availability of price-effective graphics terminals and work stations, a graphics interface between man and computers will soon be the default rather than the exception. Video games, home computers and small business computers have enhanced user expectations and have advanced the state of the art primarily with respect to 2-dimensional graphics utilities. Sophisticated (and expensive) 3-dimensional graphics systems have existed for a long time, but have mostly been reserved for dedicated special-purpose systems. The rapidly growing academic and industrial community of UNIX users needs access to more graphics tools that run on the general purpose graphics terminals typically used in this environment or on the emerging microprocessor-based workstations.

To comply with these needs and to provide an educational experience for the many students interested in computer graphics, we have developed over the last four years a modular graphics environment under the BSD 4.2 UNIX operating system. Our efforts focused on the design, representation, and rendering of mechanical parts and on the creation of purely geometrical objects in 3 and 4 dimensions. We also aim to provide some utilities to be used in the rapidly growing fields of robotics and computer vision. We now have a couple of graphics editing tools, special purpose procedural object generators, various object-modifying filter programs, and general purpose rendering tools.

Building such a large and potentially heterogeneous system at an university, where the period during which a student contributes actively to the project ranges from one to three years, has its difficulties. How do you assure that the various pieces built by individuals will hold together and do not become obsolete the moment the student leaves the university ? One approach is to produce a very detailed overall system specification at the beginning of the project, and then fill in the pieces over the years. However, in a field that moves as rapidly as the current evolution of computer graphics, this is not practical; the final system would be obsolete by the time it is completed.

A better way is to create a modular set of building blocks that can be individually developed at their own paces and that can be replaced by newer and better modules as these become available. In this approach, the only thing that needs to be defined at an early stage is the "glue" that holds everything together. In our case this was the UNIX operating system at the top level and at a lower level the *UNIGRAFIX* language. The latter is an intermediate descriptive format for the specification of objects and scenes. All modules work to and from this format. Because of its central role, we will discuss this language before we discuss some of the operational modules.

2. THE UNIGRAFIX LANGUAGE

The *UNIGRAFIX* language is a human readable, yet terse ASCII format for the description of scenes composed of 3-D polyhedral objects in boundary representation, of 2-D planar faces with arbitrary many holes, and of 1-D piecewise linear wire trains. The ASCII format makes possible easy exchange of object descriptions over electronic networks and easy modification with any text editor when an interactive graphics editor is unavailable or impractical to use. A detailed description and discussion of the *UNIGRAFIX* language was presented in 1983.[1] Only a few minor syntax changes have been made since then to make parsing more efficient and to accommodate some small language extensions.

Syntactically, a *UNIGRAFIX* file consists of statements, starting with a keyword and ending with a semicolon. Statements consist of lexical tokens, separated by commas, blanks, tabs, or newlines. The language is simple and has only ten different types of statements:

Table 1. *UNIGRAFIX* Syntax

vertices:	**v**	*ID* *x y z* ;
wires:	**w**	[*ID*] (*v1 v2 ... vn*) (...) [*colorID*] ;
faces:	**f**	[*ID*] (*v1 v2 ... vn*) (...) [*colorID*] ;
definitions:	**def**	*defID* ;
		non-def-statements
	end;	
instances:	**i**	[*ID*] (*defID* [*transformations*]) ;
arrays:	**a**	[*ID*] (*defID* [*transforms*]) *size* [*transforms*] ;
lights:	**l**	[*ID*] *intensity* [*x* *y* *z*] ;
color:	**c**	*colorID* *intensity* [*hue* *saturation*]] ;
include files:	**include**	*filename* [*transformations*] ;
comments:	**{**	[*anything* {*nesting is OK*} *but unmatched* { *or* }] }

2.1. Vertices, Wires, and Faces

Semantically, the vertices are the 'corner stones' of any *UNIGRAFIX* object description. They are described by their absolute locations in 3-D space and are then used as fix-points to define the position of 'wires' and 'faces' (edges). Rather than repeating absolute coordinates for the end-points of edges or for corners of polygons, these latter constructs simply reference previously defined vertices by their identifiers (*ID* in Table 1). A piece-wise linear wire running through 3-D space can be described with a single 'wire' statement that lists all the vertices at subsequent joints.

In the case of 'face' statements, the edge train defined by the list of vertex-IDs within a single pair of parentheses specifies a closed contour, so that it is not necessary to repeat the first vertex in a contour. Face statements with multiple groups of parentheses can be used to describe faces with several contours. Whether the contour encloses a hole or a separate patch of the face depends on its orientation and on its placement with respect to the first contour. The first contour must be an outer contour. Contours are not allowed to intersect.

2.2. Hierarchical Constructs

A building block definition capability exists with the statement pair : 'def...end'. With this construct, groups of statements can be associated with an identifier (*defID* in Table 1). Copies of these definitions can then be placed at other locations in the scene through the use of 'instance' and 'array' commands. The definitions themselves are not part of the visible scene. Instances and arrays take any homogeneous transformation for the placement of the (first) instance; in addition, the array statement needs the specification of an incremental transformation between array elements. While the definitions themselves must not be nested, the calling hierarchy can be of arbitrary depth. The permissible transformations in the above specified places are of the form:

Table 2. Transformations

–s?	*scale_ factor*	for scaling in direction of coordinate axis
–t?	*translation_ amount*	for translation along coordinate axis
–r?	*rotation_ angle*	for rotation around coordinate axis
–m?		for mirroring in direction of coordinate axis
–M3	*3x3 Matrix*	for linear 3-dimensional transformation
–M4	*4x4 Matrix*	for homogeneous 3-D transformation

The '?' should be replaced with 'x', 'y', or 'z' to denote in which direction to scale, mirror, or translate, or about which axis to rotate; as a shorthand way of specifying scaling or mirroring in 'all' dimensions, the '?' can be replaced with 'a'. When specifying a transformation in matrix form, from 1 to 9 numbers (for '-M3') or from 1 to 16 numbers (for '-M4') may be specified. The specified numbers replace the entries, by rows, in a unity matrix of degree 3 or 4, respectively. Transformations are applied to the defined object in the order given. Arguments may be integer or floating-point numbers.

2.3. Light Sources and Color

To provide shading on the faces, light sources must be specified. These can be uniform ambient lights or they can be directional. In the first case, all faces regardless of their orientation are equally illuminated. In the latter case, the brightness is determined by the scalar product between illumination vector and face normal.

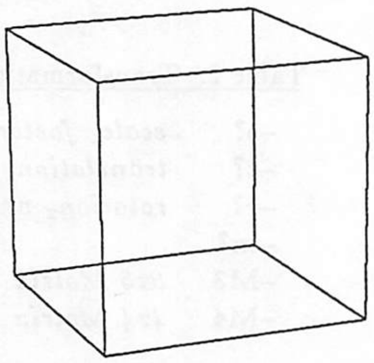
On appropriate terminals, color renderings can be produced. To specify a color, we use a double-cone model of color space. For each color, its lightness (intensity), its hue, and its saturation must be given; if the last one or two values are omitted, fully saturated colors or neutral gray surfaces are inferred, respectively. As in the case of the vertex coordinates, the lengthy color specification is not repeated for every face using that color; one simply references the corresponding *colorID* (see Table 1.).

2.4. Examples

To illustrate the correspondence between ASCII description and the defined object, we present a few simple cases. The first is the wire-frame of a cube:

Figure 1. Cube_file

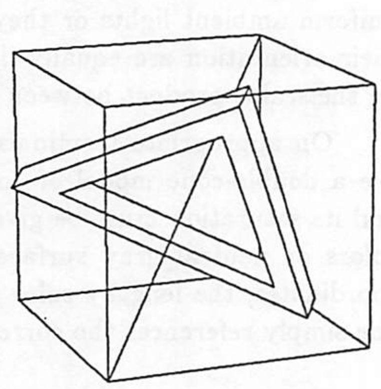
```
v XYZ     1 1 1;
v XY      1 1 -1;
v XZ      1 -1 1;
v X       1 -1 -1;
v YZ      -1 1 1;
v Y       -1 1 -1;
v Z       -1 -1 1;
v N       -1 -1 -1;
w near    ( N Y XY X N );
w far     ( Z XZ XYZ YZ Z );
w sides   ( X XZ )( Y YZ )( XY XYZ )( N Z );  {4 separate wire segments}
```



The second is an equilateral solid triangular frame embedded in a cube frame so that its symmetry axis coincides with the space diagonal of the cube:

Figure 2. Triangle_file

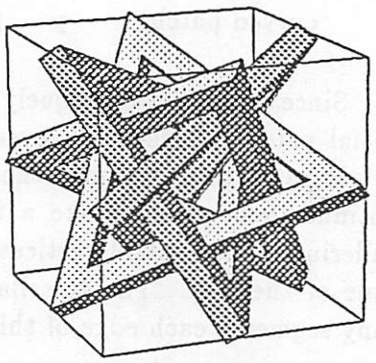
```
v  v1A     -5.4082  -0.4082   4.5917 ;
v  v1B     -4.5917   0.4082   5.4082 ;
v  v1C     -6.5917   0.4082   7.4082 ;
v  v1D     -7.4082  -0.4082   6.5917 ;
v  v2A      4.5917  -5.4082  -0.4082 ;
v  v2B      5.4082  -4.5917   0.4082 ;
v  v2C      7.4082  -6.5917   0.4082 ;
v  v2D      6.5917  -7.4082  -0.4082 ;
v  v3A     -0.4082   4.5917  -5.4082 ;
v  v3B      0.4082   5.4082  -4.5917 ;
v  v3C      0.4082   7.4082  -6.5917 ;
v  v3D     -0.4082   6.5917  -7.4082 ;
f          ( v1A  v1B  v2B  v2A );
f          ( v1C  v1D  v2D  v2C );
f          ( v2A  v2B  v3B  v3A );
f          ( v2C  v2D  v3D  v3C );
f          ( v3A  v3B  v1B  v1A );
f          ( v3C  v3D  v1D  v1C );
f          ( v1C v2C v3C )( v3B v2B v1B );  {face with triangular hole}
f          ( v3D v2D v1D )( v1A v2A v3A );  {face with triangular hole}
```



It is possible to mutually interlock four of these triangles if they are properly oriented. This can be achieved with four separate instance calls or one single array call to the triangle, which is assumed to reside in a file called 'Triangle_file'. The previously defined cube, assumed to be in a file 'Cube_file', has been scaled up by a factor of 7 to match the size of the triangles:

Figure 3. *UNIGRAFIX* Scene

```
def cube;
    include Cube_file;
end;
def triangle;
    include Triangle_file;
end;
i C ( cube  -sa 7.0 );
a T ( triangles )  4  -rz 90 ;
```



This example also demonstrates that faces with holes can be properly drawn, even if they are interlocking, without the need to cut the faces into smaller pieces. Available rendering styles include: full wire frame (Fig.1), wire frame with backface elimination, hidden lines removed (Fig.2), and shaded faces with hidden parts removed (Fig.3). Faces can be rendered with or without outlines. The latter contribute significantly to the crispness of the display when rendered on a black-and-white device.

2.5. Curved Edges and Surfaces

Over the last three years, the *UNIGRAFIX* language served its purpose well for objects and scenes composed of linear geometrical primitives. To extend the range of applicability of *UNIGRAFIX* to objects with curved edges and surfaces, we are currently experimenting with a small extension of the language, called *UniCubix*, to accommodate such elements.

We follow the approach taken in MODIF, the solids modeler developed at Tokyo University by Chiyokura et al.[2] In this system objects are entirely defined by their edges and by the borders between the curved patches, both of which can be cubic space curves. The system automatically fits patches between these borders guaranteeing 0th order continuity along edges and first order geometric (G1) continuity across the borders between patches. Thus, once a satisfactory and unambiguous method of constructing these patches has been defined, it is sufficient for the language to specify the exact shape of all edge and border curves. For cubic curves this can be done with two additional Bezier points each. This approach can of course be generalized to use more complicated construction rules for the edges. For this reason, *UniCubix* gives the curvature information in explicit edge statements (see below) rather than integrating it into the face statements. This also excludes duplicate, and possibly conflicting, specification of that information in patches sharing the same border, and it keeps the old *UNIGRAFIX* statements unchanged and makes this a straight upward extension.

Table 3. UniCubix Extensions

linear edge:	el	[*ID*] (*v1 v2*) ;
linear border:	bl	[*ID*] (*v1 v2*) ;
curved edge:	ec	[*ID*] ($v1\ v2\ b1_x\ b1_y\ b1_z\ b2_x\ b2_y\ b2_z$) ;
curved border:	bc	[*ID*] ($v1\ v2\ b1_x\ b1_y\ b1_z\ b2_x\ b2_y\ b2_z$) ;
flat face:	f	[*ID*] (*v1 v2 ... vn*) (...) [*colorID*] ; {unchanged}
curved patch:	p	[*ID*] (*v1 v2 v3* [*v4*]) [*colorID*] ;

Since patches are uniquely determined by their borders, there would be no need for a special patch statement. Nevertheless, we are planning to use a separate keyword for a curved patch to distinguish the former form flat faces. The old face statement 'f ...' will continue to be treated like a flat polygon that does not necessitate any subdivision for rendering. The control vertices of all the borders of such a patch must of course lie in the plane of the face. The tessellation of the adjacent curved patches determines into how many segments each edge of this flat face will be subdivided.

An example of a shape expressed in the *UniCubix* format is given in Figure 4. The four displayed objects are in top-to-bottom order:

- The wire frame of a polyhedral body with the same topology as the following object.
- The curved edges and borders expressed by the 'bc ... ' statements in the *UniCubix* file in Figure 4.
- The patches placed between the curved borders, approximated by meshes consisting of 25 triangular tiles; hidden lines are eliminated.
- The previous object rendered with smooth Gouraud shading and enhanced contours.

3. RENDERING

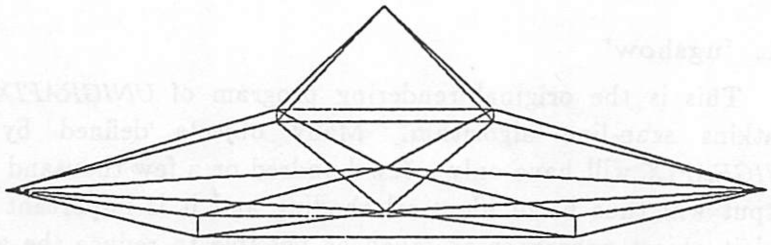
From the beginning, one of the main goals of *UNIGRAFIX* was the production of high-resolution black-and-white output of publishable quality.[1] The aim of our rendering routines was not to imitate glossy photographs of real objects, but to render the objects in a clear way in the style of an engineering drawing. In particular, it was found that the presence of outlines around each face greatly enhances the clarity of the drawing and gives it a much crisper look when rendered on black-and-white dot-raster plotters.

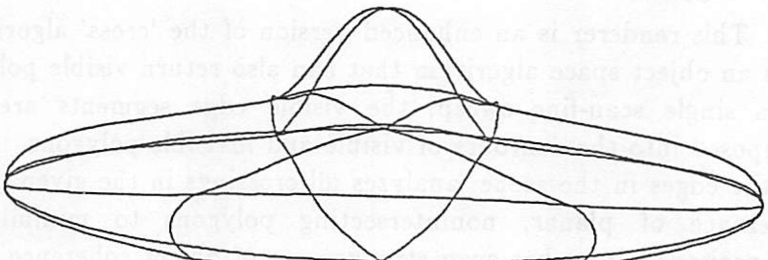
The selection of a high-resolution plotter as one of the main output devices of *UNIGRAFIX* has strongly affected the choice of the algorithms for rendering and for hidden feature removal. The widest plotter available to us measures 3 feet; a square plot of that size corresponds to about 50 million pixels. The resolution of this output medium rules out 'ray casting' as a practical rendering technique. Moreover, these plotters need the output information one line at a time in y-sorted order, thus strongly favoring a scan-line algorithm. Several high-resolution renderers based on scan-line hidden-feature removal algorithms have been developed over the last three years.

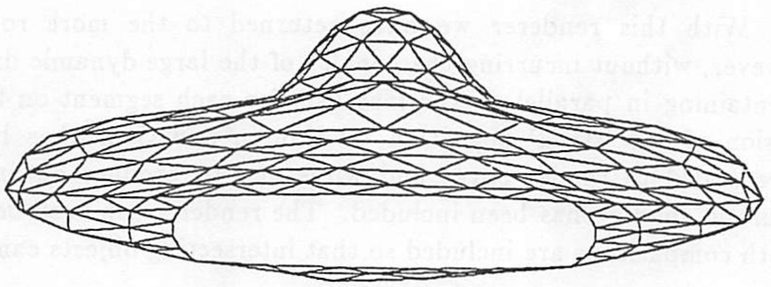
More recently, with the emergence of more interactive *UNIGRAFIX* tools, some of the earlier renderers were adapted for this new environment with different constraints. Also, in addition to these renderers that run on most typical graphics raster output devices, we have recently developed a renderer for the Silicon Graphics IRIS that makes

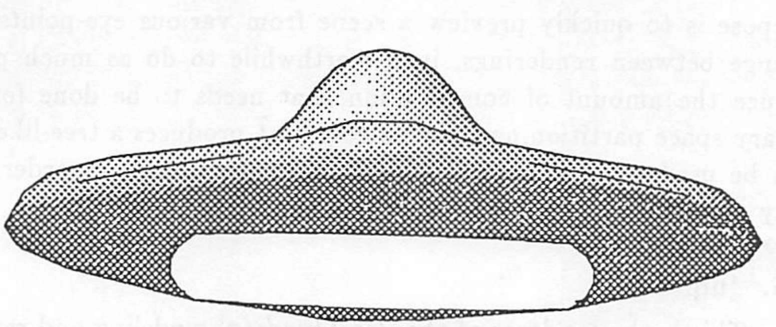
Figure 4. UniCubix Object

```
v top    0  5  0 ;
v v1A    0  2 -4 ;
v v1B    4  2  0 ;
v v1C    0  2  4 ;
v v1D   -4  2  0 ;
v v2A    7  0 -7 ;
v v2B    7  0  7 ;
v v2C   -7  0  7 ;
v v2D   -7  0 -7 ;
v v3A    0 -1 -7 ;
v v3B    7 -1  0 ;
v v3C    0 -1  7 ;
v v3D   -7 -1  0 ;
p  (v1A v1D top ) ;
p  (v1B v1A top ) ;
p  (v1C v1B top ) ;
p  (v1D v1C top ) ;
p  (v1A v1B v2A ) ;
p  (v1B v2B v2A ) ;
p  (v1B v1C v2B ) ;
p  (v1C v2C v2B ) ;
p  (v1C v1D v2C ) ;
p  (v1D v2D v2C ) ;
p  (v1D v1A v2D ) ;
p  (v1A v2A v2D ) ;
p  (v2A v3B v3A ) ;
p  (v2A v2B v3B ) ;
p  (v2B v3C v3B ) ;
p  (v2B v2C v3C ) ;
p  (v2C v3D v3C ) ;
p  (v2C v2D v3D ) ;
p  (v2D v3A v3D ) ;
p  (v2D v2A v3A ) ;
bc  (v1A v1D
   -1.4011  2.5485 -2.8635
   -2.8635  2.5485 -1.4011 );
bc  (top v1A
    0.0000  5.0000 -1.6667
   -0.0000  2.7244 -2.4990 );
bc  (v2B v2A
   10.4023 -0.0953  3.7370
   10.4023 -0.0953 -3.7370 );
bc  (v3A v2A
    2.3680 -0.8352 -7.1797
    5.4205 -0.5172 -8.7067 );
bc  (v3A v3D
   -2.6561 -1.0000 -5.5567
   -5.5567 -1.0000 -2.6561 );
. . .
```









use of the special hardware features used by this device.

In the following the various renderers will be discussed briefly.

3.1. 'ugshow'

This is the original rendering program of *UNIGRAFIX* 1, based on an enhanced Watkins scan-line algorithm. Many objects defined by the anticipated user of *UNIGRAFIX* will have only a few hundred or a few thousand vertices. Large areas in the output will thus be of identical shading and it is important to look for algorithms that exploit object coherence as much as possible to reduce the amount of computation that needs to be done. This program exploits object coherence by keeping z-ordered lists of faces for all the segments between subsequent edge crossings on the active scan line.[1]

3.2. 'ugplot'

This renderer is an enhanced version of the 'cross' algorithm by Hamlin and Gear.[3] It is an object space algorithm that can also return visible polygons at object resolution.[4] In a single scan-line sweep, the visible edge segments are determined and properly composed into the contours of visible and invisible polygons. The algorithm concentrates on the edges in the scene, analyzes all crossings in the given projection, and relies on the coherence of planar, nonintersecting polygons to minimize the number of depth comparisons. It makes even stronger use of object coherence than *ugshow*; this makes it more sensitive to small data inconsistencies.

3.3. 'ugdisp'

With this renderer we have returned to the more robust approach of *ugshow*, however, without incurring the penalty of the large dynamic data structure resulting from maintaining in parallel all the face lists for each segment on the scan-line. An enhanced version of the 'stack' algorithm by Hamlin and Gear[3] has been used. Special features were added to render edges and wires and to produce outlines around faces. Optional Gouraud shading has been included. The renderer can even be run in a mode where extra depth comparisons are included so that intersecting objects can be handled directly.[5]

3.4. 'ugpaint'

This is a renderer aimed at a display device such as the Silicon Graphics Iris. Its purpose is to quickly preview a scene from various eye-points.[6] Since the scene does not change between renderings, it is worthwhile to do as much preprocessing as possible to reduce the amount of computation that needs to be done for each displayed frame. A binary space partitioning (BSP) algorithm[7] produces a tree-like structure of the scene that can be used to determine quickly a strict back-to-front ordering of all polygons, so that they can then be rendered with a painters algorithm.

3.5. 'uq'

This is the renderer of the '*UniQuadrix*' modeling and rendering program for objects represented as the Boolean intersection of quadric and planar half-spaces. A language very similar to the *UNIGRAFIX* format is used to define the half-space boundaries by their coefficients. *UniQuadrix* uses implicit equations to represent the surfaces and boundaries of objects throughout the rendering process. This permits a scan-line based algorithm very similar to the one used in *ugshow* to quickly identify visible spans. An efficient incremental algorithm shades pixels within spans.[8]

4. SOME GENERAL UTILITIES

Because of the constraints inherent in some of the renderers discussed above and in some of the filters to be presented in the next section, *UNIGRAFIX* descriptions need sometimes be converted to "simpler" descriptions, using only a subset of the expressibility of the full *UNIGRAFIX* descriptive format. A few "filter" packages provide such services.

4.1. 'ugisect'

Most of the renderers described in the previous section rely in their hidden feature elimination algorithm on the fact that the faces are planar and do not intersect each other. Scenes that due to the construction process or due to their inherent nature consist of intersecting objects need to be preprocessed once with *ugisect*[9] to convert them to a *UNIGRAFIX* description with no intersecting faces before they can be rendered. However, this process must not generate spurious edges that would subdivide unnecessarily contiguous parts of planar faces. *Ugisect* can handle arbitrary collections of polygons. In addition, when true polyhedral solids are involved, *ugisect* can also form set theoretic operations, i.e., union, intersection, or difference of two objects.

4.2. 'ugxform'

This "filter" program makes a global transformation on a *UNIGRAFIX* file by simply transforming all vertices and instances at the top level of the scene hierarchy with the transformation specified on the command line. All other information is passed through unaltered. This utility can be used to transform the scene so that the default viewing option produces an optimal display. It can also be used to produce anisotropic scaling of vertex groups for use in other objects.

4.3. 'ugexpand'

This batch program expands instances and arrays recursively into their individual constituent parts. It produces a hierarchically flat description of vertex, wire, and face statements. This form is needed by some of the modifier programs that cannot cope with a hierarchical description. Optionally, the long and cumbersome hierarchical vertex names that may result in this process can be replaced with new terse (and meaningless) identifiers. Another important option is to merge all vertices within a distance *epsilon* of one another. This helps to avoid problems resulting from slight intersections that may be created by such near coincidences in the presence of numerical inaccuracies.

5. THE UNIGRAFIX LIBRARY

A rendering system alone is probably unsatisfactory for most users when not complemented by some tools to aid in the generation of interesting objects. Simple *UNIGRAFIX* objects can be readily created with a text editor; large, but regular objects can be generated by writing a small program (in your favorite language) that produces the required ASCII strings. The simple and terse format of the *UNIGRAFIX* language as well as its hierarchical nature make both these approaches quite practical.

The *UNIGRAFIX* library contains simple geometric primitives that are frequently used as the starting point for the generation of objects. They comprise Platonic solids in 3-D and 4-D space. In addition there are filter programs that modify these primitives or other *UNIGRAFIX* objects. Finally, several procedural generator programs have been developed that create polyhedral objects from scratch.

5.1. Generator Programs

In the following we present some examples of programs that create an *UNIGRAFIX* object description from scratch based on some user-supplied parameters or data files. These programs have typically been developed by students as course projects in graduate courses on computer graphics and solids modeling.

'ugsweep' sweeps a polygon through space with an arbitrary incremental transform between steps and produces the surface of the swept out volume.

'mkworm' creates properly mitred prismatic tube sections around piece-wise linear paths through 3-D space. These paths are read in from an 'ax'-file and can form closed loops but must not include branches.

'mktree' outputs, in the above mentioned 'ax-file' format, the joint-coordinates of a tree-like object based on the growth algorithm of Kawaguchi.[10] The shape of the generated structure can be altered by a set of command-line options to make them resemble trees, shells, or corals.

'mkstairs' creates helical staircases or ramps according to a set of parameters. Each step is an instance of a definition describing a single step or ramp segment with the proper geometry for a smooth fit.

'mkcity' is a city-sprawler that will generate a "downtown area" of a city in a random fashion. It supports three types of prism-based buildings. Other parameters control the block dimensions, number of blocks in the scene, street width, and the maximum and minimum height of the buildings.

'mkgear' produces a *UNIGRAFIX* description of gear boxes based on the specification of position and size, of gear wheels and shafts.

'mkrobot' is a generator program that reads the predefined parts of a robot arm from the file ~ug/lib/rbparts, takes the values of the various position parameters from the command line, performs some checking on the size and ranges specified, and produces a *UNIGRAFIX* description of the complete manipulator arm.

5.2. Modifier Programs

Other programs start from an existing *UNIGRAFIX* description to produce a new object, either by such processes as projection or truncation, or by modifying each individual face of the polyhedral object in some specified way:

'ugshrink' separates the faces of a polyhedron and shrinks them individually by a specified factor with respect to the face center. It can also be used to cut similarly shaped holes into faces or to produce concentric rings.

'ugfreq' subdivides triangular faces into a tessellation of similar, but smaller facets. The degree of subdivision is specified by the 'frequency' parameter.

'ugtess' is a filter that tessellates the faces of an arbitrary unigrafix object into convex polygons without creating any new vertices. An option exists to triangulate the faces instead.

'ugstar' constructs pyramid-shaped extrusions or intrusions on all faces of a polyhedron. The tip of the pyramid lies at a parameterized distance on the face-normal through the face-center.

'ugtrunc' truncates the corners of a polyhedron. New vertices are formed either in the middle of every edge or at a parameterized distance from the ends. These new vertices are then linked in a circular manner around every old vertex to form the new faces. To guarantee planar truncation faces, an approximate plane is first placed through all the vertices determined in the above manner on the edges emerging from a particular vertex; then the truncation plane is moved through the new vertex that minimizes the distance of the plane from the old vertex. Vertices with emerging edges that occupy more than a half-space (saddle points) will not get truncated.

'ugwire' creates a wire segment for every physical edge in the original polyhedron. The wire sections are disassembled at the corners and can be shortened by a specified amount. They can be used as ax-input to the mkworm program.

'ugsphere' projects all vertices radially from the origin onto a sphere of a given radius around a specified center point. This is useful to construct geodesic domes.

'ugpipe' produces ball and cylinder descriptions in the *UniQuadrix* descriptive format. It starts from standard *UNIGRAFIX* scene descriptions and converts all vertices into balls and all wire segments and face edges into cylinders. The output contains all of the quadric and planar descriptions necessary to render the object with *UniQuadrix*.

'ug4to3' projects 4-dimensional vertex coordinates into 3-dimensional space. It applies to each vertex the transformation specified. The default transformation is a parallel projection along the w-axis, i.e., simply a removal of the w-component. Face, wire, color, and light statements are passed unaltered to the output.

5.3. Modifier Pipes

In typical UNIX style, the described filter and generator programs can be piped into one-another to form very powerful scripts. The examples below show how the objects were generated as well as how they were rendered.

Figure 5.

```
cat ~ug/lib/dodeca | ugstar -h 3 |
ugtrunc -t 0.9 | ugtrunc -C -t 0.7 > f5

cat f5 illum | ugplot -ep -25 60 -100 -sa
-dw -sy 3 -sx 2.75
```

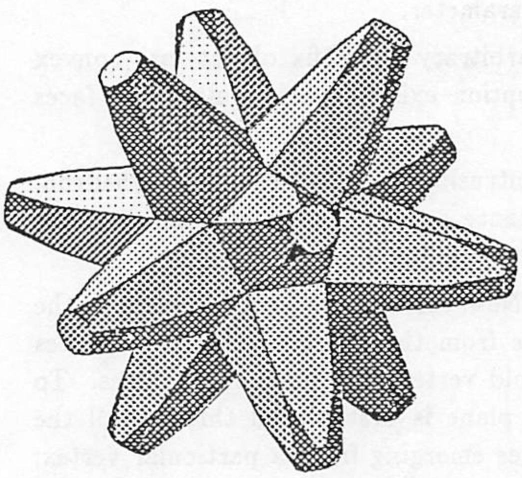


Figure 6.

```
cat wire | ugsweep -n 9 -tx 6 -rz 30 -tx
-6 -n 9 -tx -6 -rz -30 -tx 6 | ugxform -tx
6 -ty 6 | ugshrink -f 0.8 | ugsweep -tz 10
> f6

cat f6 illum | ugplot -ep -50 30 -100 -sa
-dw -sy 3 -sx 2.75
```

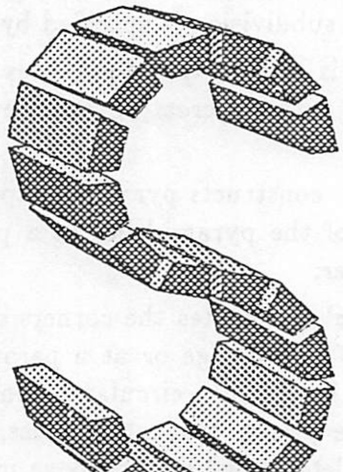


Figure 7.

```
cat ~ug/lib/icosa | ugfreq -f2 | ugsphere
| ugshrink -f0.8 -H > f7

cat f7 illum7 | ugdisp -ep -60 30 -100 -
ab -sa -sg -dw -sy 3 -sx 2.75
```

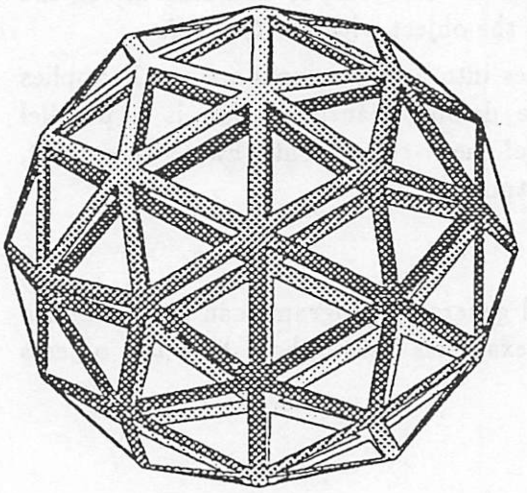
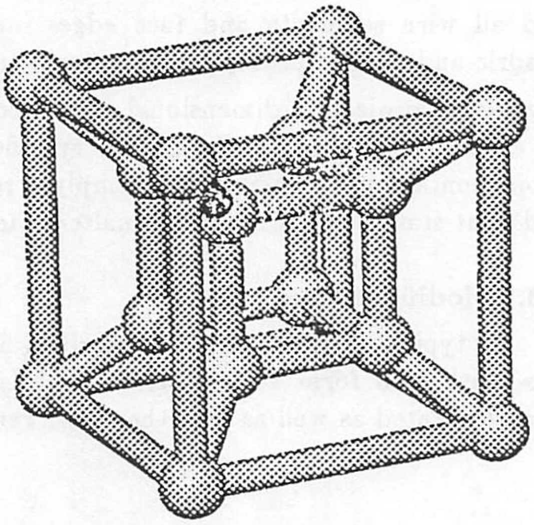


Figure 8.

```
cat ~ug/lib/D4cube | ug4to3 -ep 0 0 0 3
| ugpipe -rb 0.3 -rc 0.15 > f8

cat f8 illum8 view8 | uq
```



6. INTERACTIVE EDITING

In some sense, the *UNIGRAFIX* system construction has been started at the back end, providing the rendering programs first. Recently, the missing front end, a truly interactive editor, has been constructed in prototype form. As an intermediate step we had first created a semi-interactive environment at the shell level.

6.1. The Interactive Shell 'ugi'

This interactive environment for the display of *UNIGRAFIX* scenes provides most of the old *UNIGRAFIX* batch capabilities for scene manipulation, view specification, and display style within a homogeneous interactive framework.[11] This greatly enhances the speed and ease with which *UNIGRAFIX* scenes can be designed, viewed, and edited. The new flexible renderer, *ugdisp*, has been integrated with the new interactive environment. *ugdisp* is capable of detecting intersecting polygons and displaying them correctly, which is an important feature during the scene composition stage. Only the visible intersections are handled, so no effort is wasted on hidden areas. The rendering speed does not seem to suffer more than a factor of two for "reasonable" scenes.[5]

6.2. Creating Curved Objects with 'uci'

This is a modified version of the interactive shell *ugi*. Its main purpose is to read and interpret *UniCubix* files and to produce a *UNIGRAFIX* description for the various renderers. After the data has been read and an internal data structure representing the topology of the objects and all the control polygons of the curved edges have been constructed, the user can modify globally the "bulginess" of all curves, convert the cubic Bezier curves to piecewise linear wire approximations expressed in ordinary *UNIGRAFIX* format, or create the patches that blend together with G1 continuity and interpolate the given curves. These patches can the be approximated with polyhedral nets which again are expressed in regular *UNIGRAFIX* format.

This program also comprises an exploratory prototype of a tool to help in the creation of rounded objects of arbitrary topology. The construction starts from a polyhedral object with the same surface topology as the desired smooth object. The will result is a description of all the curved boundaries of the object in *UniCubix* format. The polyhedral object is first triangulated with *ugtess*. Averaging surface normal vectors are then determined at all vertices; they define the tangent planes at these points. Finally, cubic boundary curves are constructed that start and end in these tangent planes and replace the original straight edges of the polyhedral object.[12]

We plan to explore other rounding operations too.

6.3. The Geometric Construction Editor 'Jessie'

Jessie is an interactive tool for the creation and modification of *UNIGRAFIX* scene descriptions and individual leaf cell objects.[13] It supports a rich set of transformation operators to move and align objects or whole subtrees. This can be done visually or with constructive methods giving full geometric accuracy. *Jessie* strongly exploits the given hierarchy (if any) in the object description to gain speed in rendering and when picking objects. The first prototype is built under *SunTools* in the window package on the SUN III/160 color workstation.

7. THE FUTURE

The creation of "interesting" objects is still a time-consuming and tedious process, and we plan to create better tools to ease this task. At the level of interactive editing tools, we are particularly interested in the development of better user interfaces with more high-level commands and some built-in intelligence to recognize implicitly opportunities to produce symmetric configurations and to align objects with respect to one another. Other experiments will explore the addition of constraint systems to such editors. In time, these experiments will be extended to the *UniCubix* system.

Other extensions will concern the procedural generation of objects. As the opportunity arises, we will add more special purpose generators that create particular classes of objects. We plan also to explore language extensions that may make the construction of 3-dimensional object compilers easier for the user.

UNIGRAFIX is being made available to people for their own use and at their own risk. These programs form in no way a "turn-key system." We believe they are a basis from which others can start their own experiments, and perhaps a guide how such a system could look, once all the parts have been honed to perfection.

A new release of our existing software is planned for early 1986.

8. SUMMARY

UNIGRAFIX is a further enhancement of the UNIX environment; it makes three main contributions: First, it presents a terse ASCII-based language for the description of scenes at the object database level.

Second, it provides an efficient rendering system for high-resolution views of polyhedral objects. It produces hardcopy output in the style of engineering drawings, rather than refined displays simulating photographic renderings of real objects.

Third, it offers a collection of generator and modifier programs that make it easy for the user to create rather complex objects with a command-line pipe or with a small shell script. The currently available utilities are primarily aimed towards geometric objects such as semi-regular polyhedrons in three and four dimensions.

Overall, the Berkeley *UNIGRAFIX* tools provides the UNIX environment with long needed graphics and modeling facilities.

ACKNOWLEDGMENTS

The UNIGRAFIX system would never have come into existence without the dedicated effort of many graduate students at U.C.Berkeley. Special contributions have been made by Paul Strauss, Paul Wensley, Mark Segal, Nachshon Gal, Lucia Longhi, Ziv Gigus, and H.B. Siegel. This development has been supported by Tektronix, Inc. and by the Semiconductor Research Corporation.

References

1. C.H. Séquin and P.S. Strauss, "UNIGRAFIX," *Proc. 20th Design Automation Conf.*, pp. 374-381, Miami Beach, FL, June 1983.
2. H. Chiyokura and F. Kimura, "Design of Solids with Free-form Surfaces," *Computer Graphics (Siggraph'83 Conf. Proc.)*, vol. 17, no. 3, pp. 289-298, 1983.
3. G. Hamlin and C.W. Gear, "Raster-Scan Hidden Surface Algorithm Techniques," *Computer Graphics*, vol. 11, no. 2, pp. 206-213, Summer 1977.
4. C.H. Séquin and P.R. Wensley, "Visible Feature Return at Object Resolution," *Computer Graphics and Appl.*, vol. 5, no. 5, pp. 37-50, May 1985.
5. N. Gal, "Hidden Feature Removal and Display of Intersecting Objects in UNIGRAFIX," Master's Report, U.C. Berkeley, Jan. 1986.
6. Z. Gigus, "Binary Space Partitioning for Previewing UNIGRAFIX Scenes," Master's Report, U.C. Berkeley, Jan. 1986.
7. H. Fuchs, G.D. Abram, and E.D. Grant, "Near Real-Time Shaded Display of Rigid Objects," *Computer Graphics*, vol. 17, no. 1, pp. 65-72, July 1983.
8. G.K. Ressler, "UniQuadrix," Master's Report (UCB/CSD 85/240), U.C. Berkeley, June 1985.
9. M.G. Segal, "Partitioning Polyhedral Objects into Non-Intersecting Parts," Master's Report (in preparation), U.C. Berkeley, Spring 1986.
10. Y. Kawaguchi, "A Morphological Study of the Form of Nature," *Computer Graphics (Siggraph'82 Conf. Proc.)*, vol. 16, no. 3, pp. 223-232, 1982.
11. N. Gal, "The ugi Shell for UNIGRAFIX," Technical Report (in preparation), U.C. Berkeley, Spring 1986.
12. L. Longhi, "Interpolating Patches Between Cubic Boundaries," Master's Report, U.C. Berkeley, Dec. 1985.
13. H.B. Siegel, "Jessie: An Interactive Editor for Unigrafix," Master's Report, U.C. Berkeley, Dec. 1985.

Quaternion Splines for Animating Orientation

Tom Duff

AT&T Bell Laboratories
Murray Hill, NJ

ABSTRACT

Unit quaternions are a compact, efficient and natural way to represent the orientation of objects and cameras in 3d computer animation systems. To inbetween orientations with higher order continuity, we require spline curves confined to the unit hypersphere. B-splines may be characterized as limit curves of sequences of polygonal arcs constructed by repeated binary subdivision of the curves' control polygons. We describe an algorithm for generating the unit-quaternion analogue of these curves and prove that they have all the expected analytic properties, including a spherical version of the variation diminishing property.

CR Categories and Subject Descriptors: G.1.2 [Numerical Analysis] Spline and piecewise polynomial approximation, I.3.5 [Computer Graphics] Curve, surface, solid and object representations.

General Terms: Algorithms

Additional Keywords and Phrases: matrix splines, local splines, automated inbetweening, computer aided animation, quaternions, B-splines, Bezier curves

1. Representations of Orientation

In computer-assisted 3d animation, we often find ourselves confronted with the problem of inbetweening the orientation of objects or particularly the 'camera' that views the scene. The space of all 3d rotations is not flat, and therefore does not have a uniform coordinate system. The most familiar representations of orientation have properties that cause difficulties when generating inbetweens.

1.1. Euler Angles

In computer animation systems, orientations are probably most often represented by Euler angles or some variant of them. Euler angles describe the orientation of an object in *xyz* space by a rotation about the *z* axis followed by a rotation about the resulting transformed *x* axis followed by a rotation about the resulting *z* axis (which in general is different from the original *z* axis.) The orientation of a gyroscope flywheel mounted in a gimbal is determined by the rotations of the two gimbal rings in their bearings and of the flywheel's axis. These three angles are precisely the flywheel's Euler angles. Other choices of axes work equally well. For example, the roll-pitch-yaw coordinates used to describe aircraft and ship orientations use the *y*, *x* and *z* axes,† in order.

Euler angles have a number of peculiar properties. There are any number of representations of a particular orientation, as any integer multiple of 2π may be added to any of the angles. As noted above, Euler angles do not uniformly coordinatize orientation space. Thus, interpolation of Euler angles is not rotation invariant. That is, if two orientations are rotated by the same amount, their

† This is only true if *y* is ahead, *x* is to the right and *z* is up.

Euler angle interpolant will not be the original interpolant rotated accordingly. Furthermore, if the x rotation is 0, the final z axis will be aligned with the original z axis, causing us to loose a degree of freedom. This phenomenon is called gimbal lock. It happens to a gimballed gyroscope wheel when the flywheel's axis is aligned with the outer ring's bearing axis. Small perturbations of the gyro's orientation may have no correspondingly small change to the Euler angles. The usual result is that normally insignificant bearing friction will cause the gyro to precess.

1.2. Orthogonal Matrices

Another relatively common way to represent orientation is to specify the three vectors into which the x, y and z axes transform. These three mutually orthogonal unit vectors† are the rows of an orthogonal matrix which when multiplied on the left by a point will transform it into its rotated image. We can then interpolate orientations by interpolating the elements of the corresponding matrices. The interpolated matrices will not, in general, be orthogonal, and must be adjusted by, for example, computing $renorm(M)=M'$, where

$$M'_{0\cdot}=\frac{M_{0\cdot}}{|M_{0\cdot}|}$$

$$M'_{1\cdot}=\frac{M_{1\cdot}-(M_{1\cdot}\cdot M'_{0\cdot})M'_{0\cdot}}{|M_{1\cdot}-(M_{1\cdot}\cdot M'_{0\cdot})M'_{0\cdot}|}$$

$$M'_{2\cdot}=M'_{0\cdot}\times M'_{1\cdot}$$

Renorm re-orthogonalizes M by normalizing the first row, subtracting the appropriate multiple of the first row from the second to make them orthogonal and normalizing the result, and computing the third row as the cross product of the first two.

The problem with matrix interpolation is that the rate of change of orientation can get quite large when the columns of the interpolated matrices become small, as they can when the original matrices point in almost opposite directions. In extreme cases, the interpolated matrices may even be singular, possibly causing *renorm* to fail. For example, let

$$M_0=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},$$

and

$$M_1=\begin{bmatrix}-1&0&0\\0&-1&0\\0&0&1\end{bmatrix}.$$

and consider $M_\alpha=renorm((1-\alpha)M_0+\alpha M_1)$. M_0 is the identity matrix, and M_1 is a rotation of π about the z axis. When $\alpha<.5$, $M_\alpha=M_0$. When $\alpha>.5$, $M_\alpha=M_1$. When $\alpha=.5$, the first two rows of the argument to *renorm* are zero vectors, and *renorm* divide checks. If M_1 is perturbed slightly, the discontinuity and its associated singularity disappear, but the rate of change of M_α near $\alpha=.5$ remains large.

1.3. Quaternions

Quaternions are a non-commutative field extension of the real numbers devised by William Rowan Hamilton in the late nineteenth century specifically to represent rotations in 3-space. A quaternion Q is a 4-vector that is best though of as a generalized complex number composed of a scalar 'real part' Q_r and a 3-vector 'imaginary part' Q_v. Quaternions multiply much like complex numbers:

† Normally, we would allow an animator specify vectors of any length, and at any angle, so long as the transformed x vector has a non-zero component orthogonal to the y vector. The z vector need not be specified at all, since it is the cross product of the orthonormalized x and y vectors.

$$PQ=(P_rQ_r-P_v\cdot Q_v,\ P_rQ_v+Q_rP_v+P_v\times Q_v)\ .$$

This looks like ordinary complex multiplication except for the added cross-product term, which has the effect of making quaternion multiplication non-commutative. The multiplicative identity is (1, 0, 0, 0), and the multiplicative inverse of any quaternion is

$$Q^{-1}=\frac{(Q_r,\ -Q_v)}{Q_r^2+Q_v\cdot Q_v}$$

Any orientation in 3-space can be described as a single rotation by an angle θ about an appropriately chosen axis $(x,\ y,\ z)$. The unit quaternion

$$(\cos\frac{\theta}{2},\ \sin\frac{\theta}{2}x,\ \sin\frac{\theta}{2}y,\ \sin\frac{\theta}{2}z)$$

represents the rotation. A point P can be rotated by computing

$$Q(0,P)Q^{-1}\ .$$

Note that every orientation has two representations Q and $-Q$. $-Q$ is just the rotation by $2\pi-\theta$ about axis $(-x,\ -y,\ -z)$.

As pointed out in [2], quaternion multiplication is an efficient method of composing rotations in 3-space. A single quaternion multiplication requires 16 real multiplies and 12 adds. By contrast, orthogonal matrices require 27 multiplies and 18 adds to compose. Euler angles are hard to compose directly. Probably the easiest method is to convert them to quaternions, multiply the quaternions and convert back.

Every unit quaternion is a point on a unit hypersphere. We can interpolate between unit quaternions by interpolating along the hypersphere's geodesics. The interpolation formula is just:

$$slerp(P,\ Q,\ \alpha)=P(P^{-1}Q)^{\alpha} \tag{1}$$

The name *slerp* is due to Shoemake [11] and denotes 'spherical linear interpolation.' *Slerp* is easily seen to be coordinate independent. If we rotate P and Q by R, we get

$$slerp(RP,\ RQ,\ \alpha)=RP(P^{-1}R^{-1}RQ)^{\alpha}=RP(P^{-1}Q)^{\alpha}=R\ slerp(P,\ Q,\ \alpha)$$

This interpolant also has uniform rotational velocity. Since $P^{-1}Q$ is the rotation that carries P into Q, $(P^{-1}Q)^{1/n}$ repeated 1 to n times carries P into n uniformly spaced inbetweens.

Equation 1 is not of particular computational use. However, we can easily compute

$$P^{-1}Q=(cos\frac{\theta}{2},\ \sin\frac{\theta}{2}(x,\ y,\ z))$$

Then

$$P(P^{-1}Q)^{\alpha}=P(\cos\frac{\alpha\theta}{2},\ \sin\frac{\alpha\theta}{2}(x,\ y,\ z))$$

2. Splines

If we have more than two extreme frames and apply equation 1 to consecutive pairs of extremes, we will notice jerkiness where the segments join, caused by discontinuities in the derivative of the motion. The usual way to deal with this problem is to use some sort of piecewise curve that has higher-order continuity at the extremes. Results pertaining to spline methods in curved spaces like the surface of the unit hypersphere are hard to come by. Gabriel and Kajiya [8] describe an analytic approach generalizing the natural cubic spline to arbitrarily curved manifolds. Unfortunately, their method involves numerically solving an ordinary differential equation on a curved manifold at considerable computational expense. Shoemake [11] describes an analog of Bezier's curves confined to the surface of the hypersphere. This is not, strictly speaking, a spline formulation, but curves can be pieced together with first or second derivative continuity. His approach is strictly algebraic, apparently yielding no results about the global properties of his curves.

We will demonstrate a geometric subdivision algorithm developed by an analogy to uniform cubic B-splines that generates curves on the hypersphere that have all the analytic properties we require. To motivate the algorithm we must digress for a while and discuss spline curves in flat space.

2.1. Spline Properties

From a designer's or animator's point of view, there are a number of characteristics that a spline scheme should have:

- The curves should be coordinate system independent. That is, affine transformations of the control points should transform the curve accordingly.
- The curves should have the required number of continuous derivatives. For most applications, first or second derivative continuity is considered adequate. We say that a function is C^n when it and its first n derivatives are continuous.
- The scheme should have local control. That is, adjusting a single control point should only perturb a bounded number of curve segments. The natural cubic spline fails this test Therefore so does the Gabriel-Kajiya spline.
- The curves should be 'unsurprising;' they should not have any wiggles, loops or kinks not reflected by the control points. This is usually taken to mean that the curve should have no more zeros than the polygonal arc joining the control points. In conjunction with coordinate independence, this means that any hyperplane should intersect the curve no more often than it intersects the control polygon. This 'variation diminishing' property is usually inconsistent with the natural requirement that the curve pass through its control points. It is easy to see that the only interpolating variation diminishing curves are various parameterizations of the control polygon itself.

2.2. Matrix Splines

In many popular spline schemes, the coefficients of the polynomial segments are linear combinations of the control point values. Let D_i, $0 \le i < n$ be a sequence of control points, and let M be an s by d matrix. Then the spline curve is just

$$f(t+i) = \sum_{j=0}^{d} \sum_{k=0}^{s} x^{d-j} M_{jk} D_{i+k}, \quad 0 \le x < 1, 0 \le i \le n-d$$

When M is a 4 by 4 matrix, this is just

$$F(t+i) = [t^3\, t^2\, t\, 1]\, M \begin{bmatrix} D_i \\ D_{i+1} \\ D_{i+2} \\ D_{i+3} \end{bmatrix}$$

We will normally write $f_i(t)$ for $f(t+i)$.

The Catmull-Rom cubic spline [7], the uniform B-splines [10] and the β-splines [1], among others, are all matrix splines. For example, the uniform cubic B-spline matrix is:

$$B = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

The uniform cubic B-spline is a coordinate independent, C^2, local, variation diminishing spline.

2.3. Spline Subdivision

We can subdivide a spline curve f by splitting each piece $f_i(t)$ in two halves, replacing $f(t)$ by $g(u)$ where $u=2t$. In the cubic case, letting $f_i(t)=At^3+Bt^2+Ct+D$, we get

$$g_{2i}(u)=f_i(t/2)=A\left(\frac{t}{2}\right)^3+B\left(\frac{t}{2}\right)^2+C\frac{t}{2}+D=[t^3\ t^2\ t\ 1]S_L\begin{bmatrix}A\\B\\C\\D\end{bmatrix}$$

where

$$S_L=\frac{1}{8}\begin{bmatrix}1&0&0&0\\0&2&0&0\\0&0&4&0\\0&0&0&8\end{bmatrix}$$

and

$$g_{2i+1}(u)=f_i((t+1)/2)=A\left(\frac{t+1}{2}\right)^3+B\left(\frac{t+1}{2}\right)^2+C\frac{t+1}{2}+D=[t^3\ t^2\ t\ 1]S_R\begin{bmatrix}A\\B\\C\\D\end{bmatrix}$$

where

$$S_R=\frac{1}{8}\begin{bmatrix}1&0&0&0\\3&2&0&0\\3&4&4&0\\1&2&4&8\end{bmatrix}$$

We can compute the B-spline control points of g_{2i} and g_{2i+1} from those of f_i by multiplying by

$$B_L=B^{-1}S_LB=\frac{1}{8}\begin{bmatrix}4&4&0&0\\1&6&1&0\\0&4&4&0\\0&1&6&1\end{bmatrix}$$

and

$$B_R=B^{-1}S_RB=\frac{1}{8}\begin{bmatrix}1&6&1&0\\0&4&4&0\\0&1&6&1\\0&0&4&4\end{bmatrix}$$

Since the last three rows of B_L equal the first three rows of B_R, we can splice together the control points of all the pieces of g into a single B-spline control polygon.

We are interested in a geometric construction of the subdivided net that we can easily transfer to the surface of the hypersphere. Let D_i, $0\le i<n$ be the control points of the given cubic B-spline. To compute the control points E_i, $0\le i<2n-3$ of the subdivided spline, we calculate E_{2i} as the midpoint of line (D_i, D_{i+1}). Then, letting T_i be the midpoint of (E_{2i}, E_{2i+2}), we calculate E_{2i+1} as the midpoint of (T_i, D_{i+1}). In the limit as we repeat the subdivision process recursively, the control polygon approaches the B-spline curve itself. This subdivision scheme, due to Catmull [6], is illustrated in figure 1.

The subdivision process provides an easy proof of the variation diminishing property. The limit curve has no more zeros than the original control polygon if each stage of the subdivision process introduces no extra zeros. We can consider each dogleg segment $S_i=[E_{2i-1}, D_i, E_{2i+1}]$ of the original control polygon separately. S_i can have 0, 1 or 2 zeros (see figure 2.) If S_i has no zeros, then neither does the dogleg $S_i'=[E_{2i-1}, E_{2i}, E_{2i+1}]$ of the subdivided polygon, since it is contained entirely within the convex hull of S_i. If S_i has one zero, then so does S''. If S_i has two zeros, then S_i' must have none or two. In no case can S_i' have more zeros than S_i.

3. Spherical B-splines

The only geometric operation involved in the construction given above is bisection of line segments. The analogous construction on the surface of the unit hypersphere, replacing line segment bisection with great circle arc bisection, yields a spline curve with many of the properties of B-splines:

- The spline is invariant under affine transformations of the hypersphere, since the bisection operation is.
- The curves are C^2. Since the surface of the hypersphere is locally flat, the curves inherit all their local properties from the B-splines.
- The spline has local control, since the construction is local.
- The curves are 'spherically variation diminishing.' That is, the curve has no more intersections with a hyperplane through the center of the hypersphere than its curvilinear control polygon. The proof of the flat case transfers immediately to the hypersphere, since it depends only on continuity of the control polygons, on points of the subdivided polygon being convex combinations of previous points and on line segments intersecting hyperplanes at most once. A great circle arc can intersect a hyperplane through the origin twice, but the arcs of our dogleg segments are at most half a full circle and can therefore only have one intersection.

3.1. Implementation

We can evaluate the spline function to any desired precision using the subdivision method directly. Since the hypersphere is locally flat, we won't go far wrong if we subdivide down to a certain level and then evaluate the ordinary B-spline in Euclidean space directly, projecting the resulting non-unit quaternion back onto the hypersphere. Another alternative is to subdivide down to some level and then use *slerp* to interpolate along the control polygon. If we are evaluating the function at a sequence of points (say the frames of an animation), we can cache the subdivisions to avoid repeated calculation.

It is implicit in our formulation that the control points D_i are at equal parameter intervals. This is unrealistic in practice. One way to deal with the problem, due to Loren Carpenter [4], is to use a cubic B-spline function that maps the uniform parameter into time. If the time values are monotonic so is the B-spline, since B-splines are variation diminishing. The inverse of this spline is a C^2 function that maps from time to the parameter space. Therefore the composition of the inverse time spline and the spherical B-spline is still C^2.

There are a couple of details that any implementation of this scheme will have to iron out. First, if $D_{i+1}=-D_i$ for some i, there is no unique geodesic joining the two points. The subdivision method can never generate $E_{i+1}=E_i$, so this is only a problem at the first level of subdivision. Since D_i and $-D_i$ represent the same orientation, this case need never come up. Secondly, between any two points P and Q there are two great circle arcs. One of them goes around the long way, going from P through $-Q$ and $-P$ before arriving at Q. When we bisect PQ, we should pick the bisector of the short arc. This is easy enough, the correct bisector is just $(P+Q)/|P+Q|$.

4. Conclusions

Unit quaternions are the natural representation of orientation in 3-space. Composing orientations by quaternion multiplication is easier than for any other known representation.† We can generate natural, uniform inbetween of pairs of orientations represented as unit quaternions by interpolating along geodesics of the unit hypersphere.

We have developed a unit quaternion spline formulation by analogy with a cubic B-spline subdivision method. The curves generated are rotation independent, C^2, spherically variation diminishing

†We haven't mentioned rotations represented as exponentials of skew-symmetric matrices [3]. I am aware of no animation system that uses them, and can see no good reason to adopt them.

piecewise splines. The algorithm is in use in a computer animation system in development at Bell Labs and has already been used in actual film production.

Other curve subdivision schemes are described in [5] and [9]. Lane and Carpenter's Bezier curve subdivision formula, when cast in geometric terms yields a proof similar to ours of the Bezier curves' variation diminishing property. Carrying that method onto the surface of the hypersphere yields a spherically variation diminishing curve similar to Shoemake's, but differing in detail.

Our proof of the spherical variation diminishing property answers Shoemake's [11] question of the existence of simple unit quaternion curves with provable variational properties. We would like to find a direct analytic description of the curves our method generates.

5. Acknowledgements

Alvy Ray Smith suggested investigating quaternion splines. Andrew Hume prodded me to get the code working.

6. References

[1] Brian Barsky, *The Beta-spline: A Local Representation Based On Shape Parameters and Fundamental Geometric Measures*, PhD dissertation, Department of Computer Science, University of Utah, Salt Lake City, December 1981

[2] M. Beeler, W. Gosper et al, *Hakmem*, MIT AI Lab memo #239, 1973

[3] R. W. Brockett, "Robotic Manipulators and the Product of Exponentials Formula," *Proceedings of the MTNS-83 International Symposium*, Beer Sheva, Israel, June 1983, pp. 120-129

[4] Loren Carpenter, personal communication, 1981

[5] Edwin Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Ph.D. dissertation, Department of Computer Science, University of Utah, Salt Lake City, December 1974

[6] Edwin Catmull, personal communication, 1977

[7] Edwin Catmull and Raphael Rom, "A Class of Local Interpolating Splines," *Computer Aided Geometric Design*, edited by Robert E. Barnhill and Richard F. Riesenfeld, Academic Press, San Francisco, 1974, pp. 317-326

[8] Steven A. Gabriel and James T. Kajiya, "Spline Interpolation in Curved Space," SIGGRAPH '85 Tutorial Notes, July 1985

[9] Jeffrey M. Lane, Loren C. Carpenter, Turner Whitted and James F. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," FICommunications of the ACM, *Volume 23, Number 1, January 1980, pp. 23-34*

[10] I. J. Schonberg, Cardinal Spline Interpolation, CBMS Vol. 12, SIAM, 1973

[11] Ken Shoemake, "Animating Rotations with Quaternion Curves," SIGGRAPH '85 Proceedings, July 1985

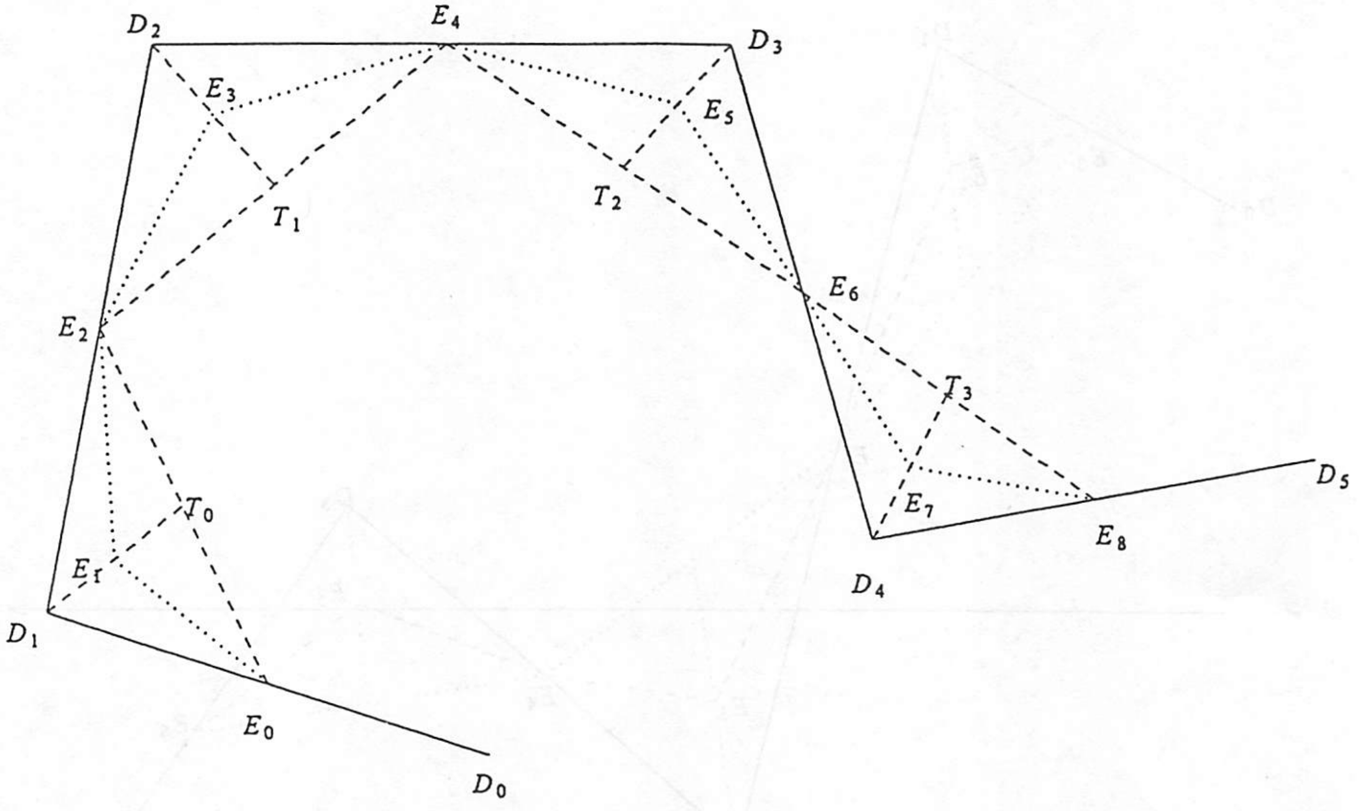


Figure 1 – Geometric B-spline subdivision
The solid lines with endpoints D_i form the original control polygon. The Dotted lines with endpoints E_i form the subdivided control polygon. The dashed lines are construction lines and T_i intermediate points in the construction.

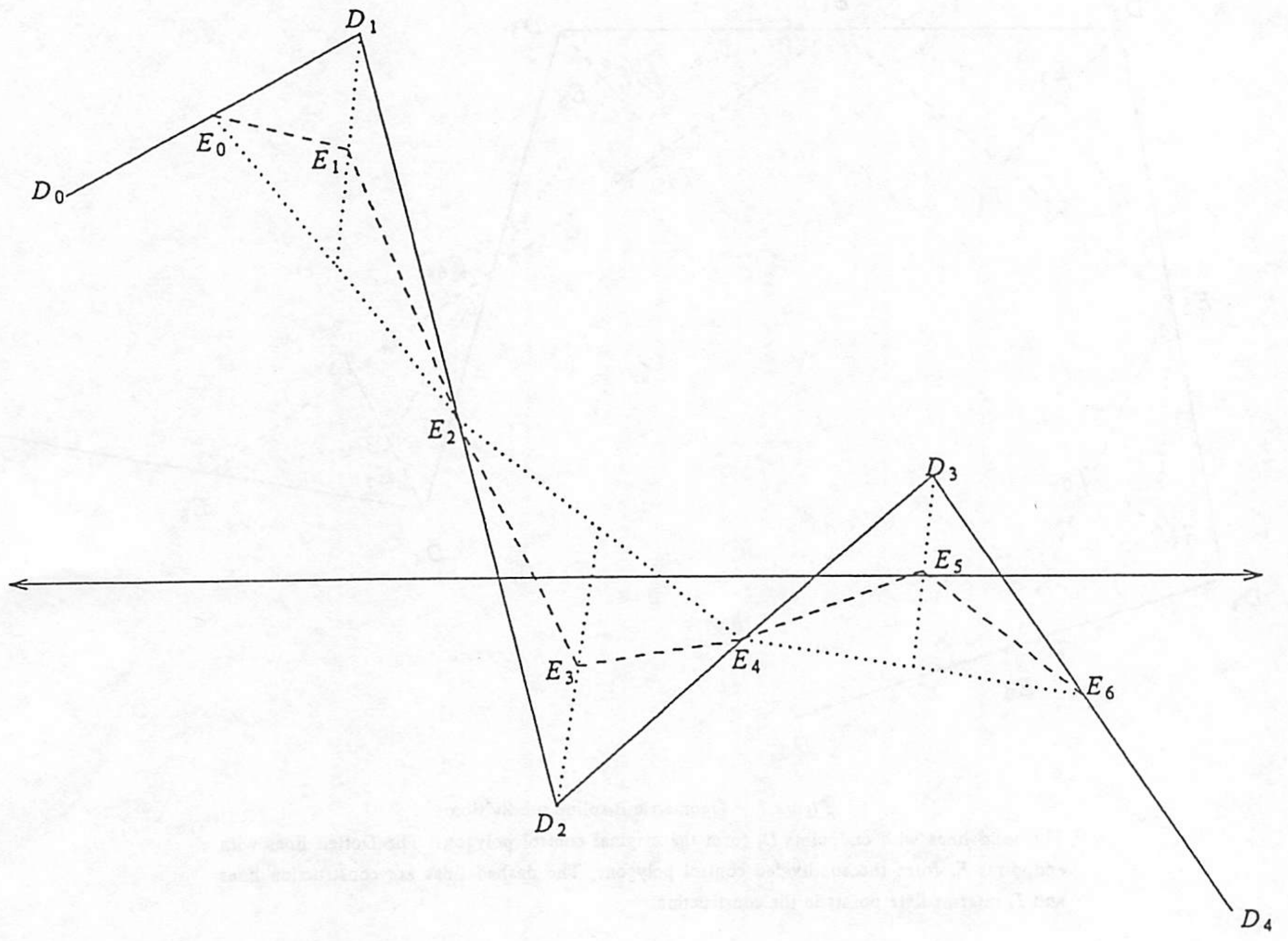


Figure 2 – Three Doglegs
The solid horizontal line is the x axis. Dogleg $[E_0, D_1, E_2]$ has no intersections with the x axis, $[E_2, D_2, E_4]$ has one and $[E_4, D_3, E_6]$ has two.

Scattered Thoughts on Color

Roy Hall
Wavefront Technologies, Inc.
1421 State Street, Suite H
Santa Barbara, CA 93101

1. Introduction

Literature in computer graphics routinely examines visible surface techniques, antialiasing, and illumination models for determining the color of surfaces. However, the translation of these colors to the image that is displayed on the monitor is seldom given much attention. This discussion addresses the practical application of color science to computer graphics for color computation and display.

The relationship can be developed in three stages. The first is understanding colorimetry as a perceptually based study and relating it to the three phosphor additive color reproduction method used in rgb monitors. The second stage is exploring the features and limitations of the monitor and how this piece of equipment can be best used for display. And finally, the third stage is reviewing the process of using spectral information for light sources and materials in the application of the illumination model so that the correct color is computed.

The information presented here is a background tutorial and a review of current work I have been doing. It is by no means an authoritative reference, but is intended as a discussion of the process of addressing color issues, and describes solution attempts that have produced improvements.

2. Colorimetry and the RGB Monitor

Colorimetry is a perceptual science, that is, it studies and attempts quantify how the human visual system perceives color. This study of perception has resulted in an empirically and statistically derived system of standards that can be applied to computer graphics with desirable and predictable results.

2.1. A Distilled Look at Colorimetry

Imagine that we have an experimental setup where a test light can shine on one half of a viewers field and a set of control lights shine on the other half (figure 1, slide 1) The control lights are sources at 1nm wavelength increments throughout the visible spectrum (roughly 380nm to 770nm) with an intensity control for each source. When a test light is shone on half the field, the viewer matches the color of the test field by adjusting the intensities of the control lights the illuminate the other half of the field.

For most colors, there are limitless different combinations of control source intensities that will provide a match. This leads us to conclude the visual system does not have a receptor for each wavelength. Studies of the eye show there are three types of color receptors called cones, and thus we expect that if we could selectively excite the types of cones it would be possible to produce any color sensation. Empirical studies showed this was true and that three correctly selected control lights in the short, medium, and long wavelength range most test colors could be matched (figure 2, slide 2).

The possibility of matching a test color with some combination of intensities of three control lights becomes useful if the required intensities can be predicted given the spectral curve of the test color. The test color sensation is a summation of sensations produced by the intensity at every wavelength in the visible range. The spectral curve for the test light is a graph of the intensity as a function of wavelength. If the required control light intensities to match every wavelength is known, then the spectral curve for the test light can be multiplied by the curves that match the control lights to every wavelength, and the areas under the resulting three curves are the required intensities of the control lights to produce a perceptual match the test light.

To determine the curves that match the control lights to any given wavelength, three control lights are chosen, say 444nm, 526nm, and 645nm, and these are matched to test lights at 1nm increments throughout the visible range. In attempting to match the single wavelength test lights only the 444nm, 526nm, and 645nm lights can be matched exactly. The other test lights can be matched in hue by a combination of two of the control lights, but the observer needs to subtract some of the third control light color in order to match saturation. Obviously, negative color cannot be added to the control side, but with some quick algebra, adding some of the third control color to the test side should produce the same result, (figure 3, slide 3). This experiment needs to be performed with a representative sample of observers and some type of statistical analysis performed to generate the average human response.

Once the matching has been performed, the resultant graph of the matching curves for the 444nm, 526nm, and 645nm curves can be plotted, figure 4, (slide 4). A more commonly used representation is created by a 2-D projection of a plot of these values in 3-space. Each orthogonal axis represents one of the control sources, L (low wavelength), M (medium wavelength), and H (high wavelength). For each wavelength the vector LMH is plotted and extended until it intersects the plane L+M+H = 1.0. The coordinates of this intersection, LMH' are found by dividing each of L, M, and H by the sum L+M+H, figure 5. The curve defined by the intersection points of the vectors and the plane is then projected into 2D by looking down the -L axis, figure 6 (slide5). The representation presents a plot of hue and saturation called the chromaticity diagram. The triangle created by intersections of the three axis and the plane describes all of the colors that can be reproduced by the three primaries. The curve describes the pure spectral colors and all of the visible colors are represented by the interior of the curve.

The primaries that are used by RGB monitors vary. To make the information described above useful, it must be cast in terms of the primaries being used for color reproduction. Given the spectral curves for the RGB phosphors, we can determine the intensities of LHM required to match each phosphor, i.e.:

$$
\begin{aligned}
R &= aL + bM + cH \\
G &= dL + eM + fH \\
B &= gL + hM + iH
\end{aligned}
$$

Which can be written in matrix form as:

$$
[L\ M\ H] \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} = [L\ M\ H] \begin{bmatrix} & & \\ & T & \\ & & \end{bmatrix} = [R\ G\ B]
$$

Using this relationship, the matching curves can be transformed for any set of primaries, figure 7 (slides 6 and 7).

The CIE (Commission Internationale d'Eclairage) set up a standard hypothetical primary set, XYZ, that resulted in all visible colors being in the positive octant and whose integrated matching functions are of equal area. These primaries are hypothetical in that they are a mathematical abstraction for convenience that do not exist in reality, figures 8 and 9 (slides 8 and 9). The implication of equal area is that a flat spectral curve (equal intensity at all wavelengths) is represented by equal XYZ values. Additionally, the selection of XYZ primaries was made so that the Y matching curve was identical to the perceived intensity function (luminous efficiency function).

This has been a very simplified and idealized discussion of colorimetry. In depth information can be obtained from many excellent references such as [CORN70], [HUNT75], [JUDD75], and [MEYE83].

2.2. The Relationship of CIEXYZ to your RGB Monitor

Proper display of colors on a monitor requires the spectral curve or the CIEXYZ chromaticity of a color be known, and the transformation from CIEXYZ to the RGB primaries of the monitor be known. If the spectral curve of the color is known it can be sampled into CIEXYZ by multiplying the curve by the CIEXYZ tristimulous matching functions and integrating the resulting curves. The values of the CIEXYZ tristimulus matching functions are available in many references, [JUDD75],[SIGG83a] and will not be repeated here. An alternate method is to transform the CIEXYZ tristimulous functions into the RGB matching functions for the monitor and perform the sampling with these curves resulting in the RGB values directly (slide 10). In either case, the transformation from CIEXYZ to RGB is required.

It may be noted that 2-degree and 10-degree CIEXYZ tristimulous matching curves are available. This refers to the view angle for the color patches during the color matching experiments. For graphics applications the 2-degree curves should be used as the images generally contain very small patches of individual colors.

The generation of the CIEXYZ to RGB matrix requires chromaticity data for the monitor phosphors. This can usually be obtained from the monitor manufacturer. There is equipment available for measuring the chromaticity of a light source (Minolta makes a luminance chromaticity meter, model CL-100, available for approx $1200). Given the chromaticities of the monitor phosphors and the white point for the monitor, the following relationships exist:

red phosphor:	rx	ry	rz=1-rx-ry
green phosphor:	gx	gy	gz=1-gx-gy
blue phosphor:	bx	by	bz=1-bx-by
white point:	wx	wy	wz=1-wx-wz

$$[R\ G\ B]\begin{bmatrix} Sr(rx) & Sr(ry) & Sr(rz) \\ Sg(gx) & Sg(gy) & Sg(gz) \\ Sb(bx) & Sb(by) & Sb(bz) \end{bmatrix} = [X\ Y\ Z]$$

In this relationship, Sr, Sg, and Sb are scale factors for red, green, and blue, that set the correct white point for the transformation. Select RGB and XYZ values corresponding to the white point and then solve for Sr,Sg, and Sb. Note the RGB white point is 1,1,1; and the XYZ white point is wx, wy, and wz. I have find it useful to use the NTSC standard white point chromaticity of wx=.310,wy=.316 (I set my monitors to this) and to scale the XYZ white so Y=1.0. The Macbeth ColorChecker chart is useful in assuring that the matrix has been properly generated [MCCA76] (slide 11). This chart with a data sheet giving the chromaticity of the color blocks is available at most photographic supply outlets. This is a standard photographic reference chart and a monitor display of this is a useful calibration tool when shooting film from the monitor.

The RGB to CIEXYZ matrices are also useful for displaying images generated for a given monitor on a monitor with different phosphors. This requires a transformation of the image data from RGB of the first monitor to R'G'B' of the second monitor. This transformation is expressed as:

$$[R\ G\ B] \begin{bmatrix} \text{RGB} \\ \text{to} \\ \text{XYZ} \end{bmatrix} \begin{bmatrix} \text{R'G'B'} \\ \text{to} \\ \text{XYZ} \end{bmatrix}^{-1} = [\text{R'G'B'}]$$

3. Care and Feeding of the RGB Monitor

The high quality RGB monitor has an abundance of possible adjustments. Attempts are made to correctly align monitors at the factory, but through time the electronics age and drift, phosphors burn and loose their original brightness, and a variety of other things occur which change the characteristics of the monitor. This section describes these characteristics and either adjusting them or working around them.

3.1. Setting the Color Balance

Setting the color balance is setting the white point for the monitor. Most high quality monitors have color offset and gain adjustments for each of the guns. The offset is used to set the white point for a very low intensity display while the gains set the white point for full intensity display. To check or reset these values it is necessary to have a white point reference. This reference can be either a properly illuminated comparison card or a chromaticity measuring device.

Follow the directions in the monitor manual for adjustment. The monitor should be turned on for several hours, preferably days, to allow for temperature and electrical component stabilization; and be in the installed position, so that the magnetic effects of the environment are stable when adjustments are made. The position of the monitor is really critical for gun convergence as opposed to color calibration. Normally the low intensity white point is set first, then the high intensity white point. This may require a few iterations as these are interdependent.

A properly illuminated white card is easier to use for adjustment because the required adjustment can be immediately observed. To use a chromaticity meter it helps to write a program that computes the difference between the measured chromaticity and the desired white point and transforms this into an RGB difference describing what adjustment is required.

If it is not possible to adjust the monitor, the next best alternatives are to either measure the white point and base all monitor transformations on that white point, or to load video lookup tables that correct the white point to the desired chromaticity. Using video lookup tables is the less desirable option because it tends to decrease the effective color resolution of the display.

3.2. Monitor Gamma, Contrast, and Brightness

Monitors normally have a nonlinear response curve, that is, the intensity displayed is not linearly proportional to the input signal level. The monitor 'gamma' is a measure of this nonlinearity and relates displayed intensity to input signal by the relationship:

$$\text{displayed intensity} = \text{input signal}^{\text{gamma}}$$

where displayed intensity and input signal are both normalized from 0 to 1. Typical monitor response is plotted in figure 10. The value for gamma normally ranges from 2.0 to 3.0 and can be established for a particular monitor by measuring the brightness of gray fields displayed in the monitor (usually the gamma is nearly for the three primary guns although it may differ). The standard NTSC gamma is 2.2.

Displaying colors without properly accounting for the gamma of the monitor will result in both a shift in contrast and a shift in chromaticity. Using a gamma that is too high will decrease the contrast and shift chromaticities towards the white point. Using a gamma that is too low will increase the contrast and shift chromaticities towards the primaries. A graph of the chromaticity shifts for the macbeth colors as a result of improper gamma correction is shown in figure 11, (slide 12).

The contrast and brightness should be set according to manufactures specifications. Usually there are preset contrast and brightness controls that can be overridden by contol knobs on the front panel of the monitor. The preset values should ALWAYS be used for image viewing so that there is a consistency between what is in image files, what is displayed, and what is recorded to film or videotape.

3.3. Using the Display System to Maximum Potential

The display system normally consists of a frame buffer and monitor. The frame buffer is 24-bits deep divided into red, green, and blue channels of 8-bits each. The signal from each of these channels is sent through a video lookup table and is then converted into an analog signal that is sent to the monitor gun, figure 12. The video lookup tables are typically 8-bits in, 8-bits out into an 8-bit digital to analog converter. The result of this is that there are 256 discrete linearly spaced voltage levels that can be sent to the monitor.

The human visual system has a logarithmic response curve. The apparent brightness of in intensity step is relative to the average intensity of the patches on either side of the step. A graph of this response is shown in figure 13. It is apparent that the visual system has greater sensitivity in the low intensity range, and it would be desirable to concentrate the image intensity resolution in this range.

It is common to store images as linear intensity RGB values. Video lookup tables are loaded into the frame buffer to correct for nonlinearity of the monitor at display time. The video lookup table values for a monitor gamma of 2.6 are plotted in figure 14. Note the steepness of the gamma table curve in the low intensity region. This maps small changes in the image value into large changes in the signal sent to the monitor thereby decreasing the effective resolution of the display system in the low intensity region. The use of video lookup tables to correct for monitor nonlinearity results in a loss of resolution in the intensity range where the human visual system is most sensitive.

Note that if the curve of perceived brightness is mapped through the response curve of the monitor the result approximates a linear curve. The maximum color intensity resolution of the display system can be realized only if the image is gamma corrected before conversion into the byte value that are stored in the image file. Thus the video lookup tables can be bypassed and the full range of frame buffer values, 0-255 by steps of 1, will reach the monitor.

Storing the images in a gamma corrected form can cause problems if the images are to be viewed on a variety of monitors with different response curves. The compromise solution that I use is to store images with standard NTSC gamma of 2.2 applied and to video lookup tables that correct the monitor to this gamma. This puts the images in the proper format for video recording with no video lookup tables, also, the mapping for most monitors is then nearly a linear ramp video lookup table thus minimizing the loss of resolution.

4. Spectral Computation of Color

Although there has been a great deal of attention given to illumination models in computer graphics, arbitrary RGB triplets are normally plugged into these models in the hopes of generating realistic images. After going to the trouble of understanding and calibrating the color reproduction process, combined with the trouble of generating these complex illumination models, it seems reasonable to use the spectral curves of real materials for determining the correct color. This section addresses some of the considerations surrounding the use of measured spectral curves.

4.1. Spectral Curves for Materials and Lights

Spectral curves for a wide range of materials are available in a number materials handbooks. The material studies from Purdue University [PURD70] are an excellent resource for these curves. Tables of spectral curves for standard illuminants are available in reference texts [JUDD75] and manufacturers of bulbs and fixtures can be consulted for information on specific types of sources.

The sampling curves have been developed in a previous section that can be used to match the RGB values for any of these curves.

4.2. RGB vs XYZ vs ANYTHING ELSE

Illumination models are generally expressed without specific reference to the wavelength of the light used. It is implicit that the calculation is repeated for every wavelength, which is generally interpreted as repeated for each of the three primary colors; red, green, and blue. While illumination calculations are normally performed this way, it is questionable whether this is the best method.

Consider a simple reflection of a light source from a perfectly polished surface, the energy from the source is either absorbed or reflected in the mirror direction. The spectral curve of the source describes the energy reaching the surface at each wavelength. The spectral reflectivity curve of the material describes the percentage of the incident energy at each wavelength that will be reflected, the rest is absorbed (fresnel effects and other considerations are ignored in this simple example). The spectral curve for the reflected light is obtained by multiplying the spectral curve of the source by the spectral reflectivity of the surface on a wavelength-by-wavelength basis.

Information is lost when spectral curves are reduced to a representation by three numbers, such as RGB. The concern is whether reducing source and material spectral curves to RGB values before the multiplication results in a RGB value for the reflected light that is the same as reducing the spectral curve of the reflected light to RGB values. If the results are considerably different this raises question as to whether the RGB notation is the appropriate choice for color computation.

Consider a perfect mirror surface, that is, a surface that reflects all of the incident light at every wavelength with no absorption. The spectral reflectivity curve for this material has a value of 1.0 at all wavelengths. Any light source, such as a D6500 source will be reflected with an unchanged spectral curve, the reflected color will be identical to the incident color. Using the primaries and white point for my monitor, the RGB for the perfect mirror is

1.103,0.970,0.822 and the RGB for the D6500 light source is 0.956,1.019,0.898. The resultant calculated RGB is 1.055,0.988,0.738 which differs a great deal from the original light source color.

A possible solution is to use the CIEXYZ values for computation and to transform to RGB after computation. Remember that the CIEXYZ curves are normalized to equal area and can therefore be scaled so that the perfect mirror samples to an XYZ of 1,1,1. An approach that I use is to define a number of simple box sampling and reconstruction functions. The sampling functions reduce the spectral curves to a number of values that can be used for computation, and then spectral curves can be reconstructed and matched to RGB or CIEXYZ values. I determine the bounds of the sampling and reconstruction functions using a non-rigorous statistical analysis of a number of material and light source spectral curves (this is just an approach, not a solution to the problem). Details and discussion of this may be found in [HALL83a][HALL83b].

4.3. An Example of the Problem

A test case that I used to demonstrate the problem was light passing through two blocks of filter materials with very extreme spectral curves, figure 15. The sharp cutoff frequencies were carefully selected for a worst case condition in both the RGB and CIEXYZ solution. The results of computing images at 1nm wavelength increments, using 9 sample functions, in RGB, and in CIEXYZ are shown in slide 13. Note that the 9 sample functions match the 1nm increment control image quite closely while RGB and CIEXYZ are very different.

From experience I have found that using the CIEXYZ sampling for computation actually produces very good results with most naturally occuring materials and light sources. The consideration of laser light, photographic filters, gas discharge lights and other man-made sources and materials can require many more than 3 samples for reasonable results.

5. Conclusion

For the color quality of the generated images to keep pace with the advances in image generation technique, it is necessary to seriously approach the color issue. Tacitly relying on color computations in RGB space will not be sufficient in color critical applications of computer generated imagery. This discussion has highlighted problems and discussed approaches towards discovering solutions to these problems. It is apparent that much work remains to be done in this area.

6. REFERENCES

CORN77	Cornsweet, T.N., _Visual Perception_, Academic Press, New York, 1970.
HALL83a	Hall, Roy, "A Methodology for Realistic Image Synthesis," Masters Thesis, Cornell University, Ithaca, 1983.
HALL83b	Hall, Roy and Donald Greenberg, "A Testbed for Realistic Image Synthesis," IEEE Computer Graphics and Applications, vol.3, no.8, pp.10-20, 1983.
HUNT75	Hunt, R.W.G., _The Reproduction of Color_, Third Edition, New York, John Wiley and Sons, 1975.
JUDD75	Judd, D.B. and Wyszecki, G., _Color in Business, Science, and Industry_, New York, John Wiley and sons, 1975.
MCCA76	McCamy, C.S., H.Marcus, J.G.Davidson, "A Color Rendition Chart," _Journal Of Applied Photographic Engineering_, vol.11, no.3, 1976.
MEYE83	Meyer, Gary, "Colorimetry and Computer Graphics," Program of Computer Graphics Report No. 83-1, Cornell University, Ithaca, New York, 1983
PURD70	Purdue University, _Thermophysical Properties of Matter_, 1970
SIGG83a	"Color Perception," SIGGRAPH tutorial course notes, july 1983.
SIGG83b	"Introduction to Raster Graphics," SIGGRAPH tutorial course notes, july 1983.

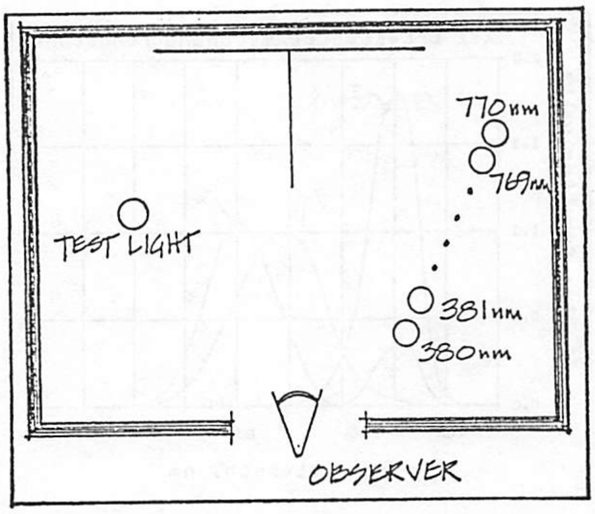


Figure 1 Color matching experiment with control sources at 1nm

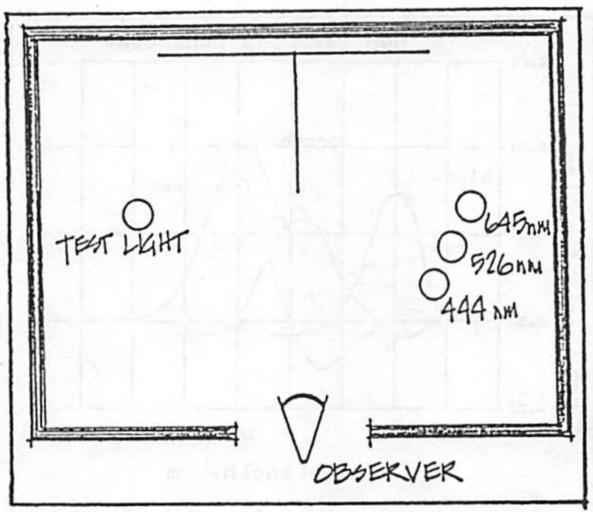


Figure 2 Color matching experiment with 444nm, 526nm, and 645nm lights

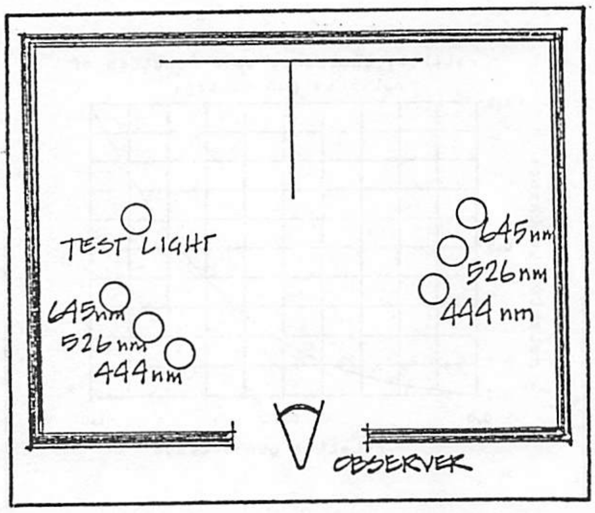


Figure 3 Color matching experiment with control sources for negative color

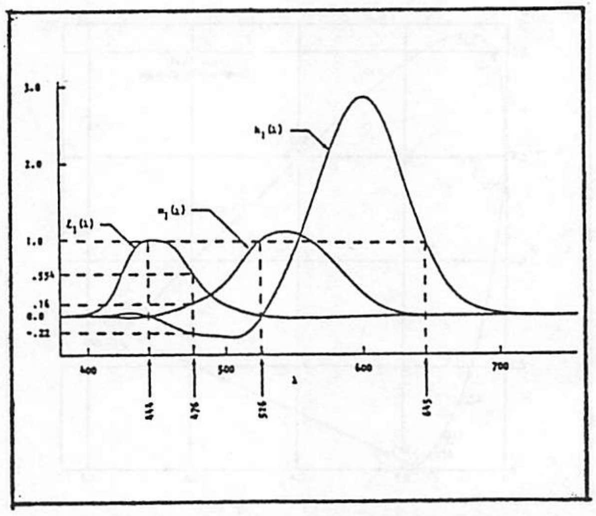


Figure 4 Graph of the color matching curves for 444nm, 526nm, and 645nm lights (from MEYE83)

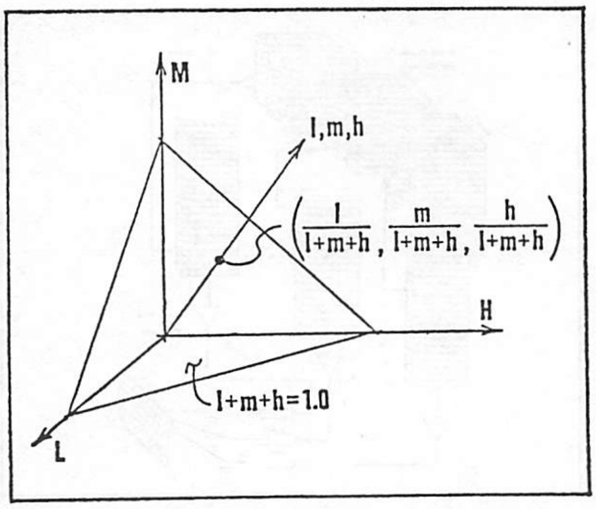

M
l,m,h
$\left(\frac{l}{l+m+h}, \frac{m}{l+m+h}, \frac{h}{l+m+h}\right)$
H
l+m+h=1.0
L

Figure 5 Projection of the chromaticity diagram

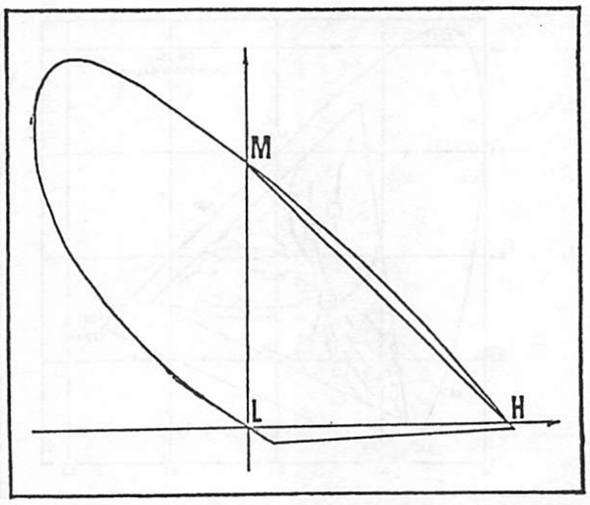


Figure 6 Chromaticity diagram for 444nm, 526nm, and 645nm primaries

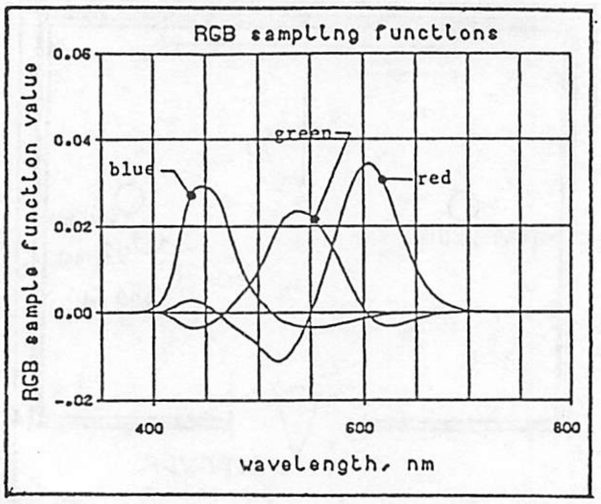


Figure 7 Matching curves for typical monitor primaries

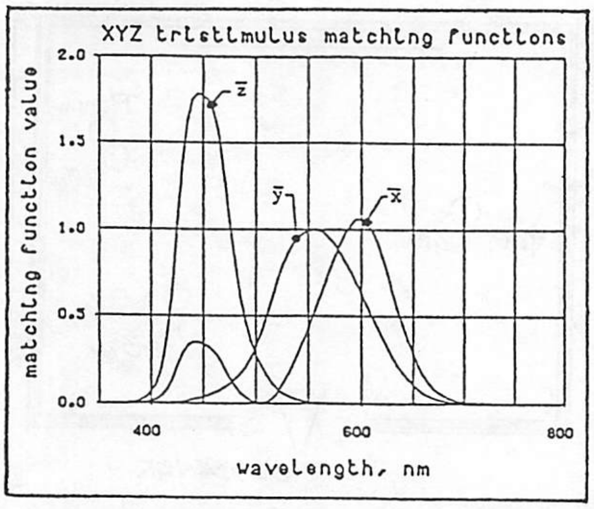


Figure 8 CIEXYZ matching curves

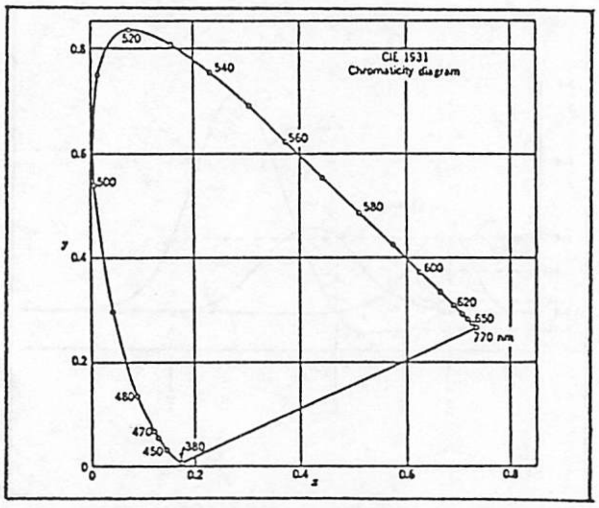


Figure 9 CIEXYZ chromaticity diagram

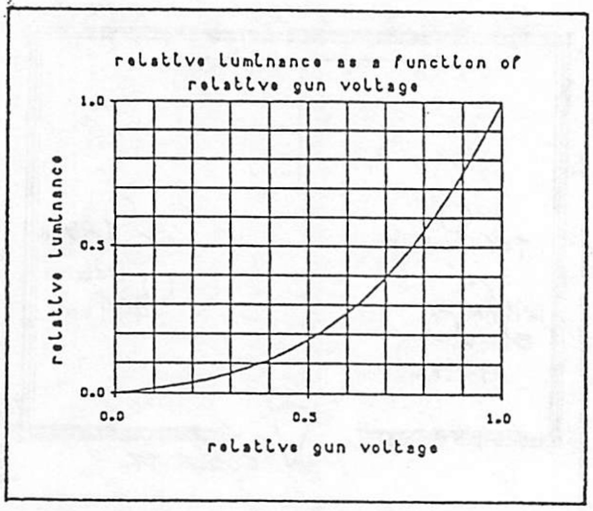


Figure 10 Typical Monitor response curve

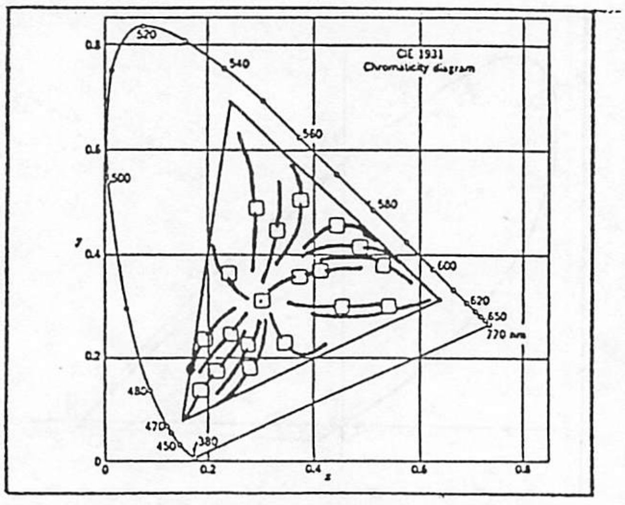


Figure 11 Shift of Macbeth color chart colors as a function of incorrect display gamma

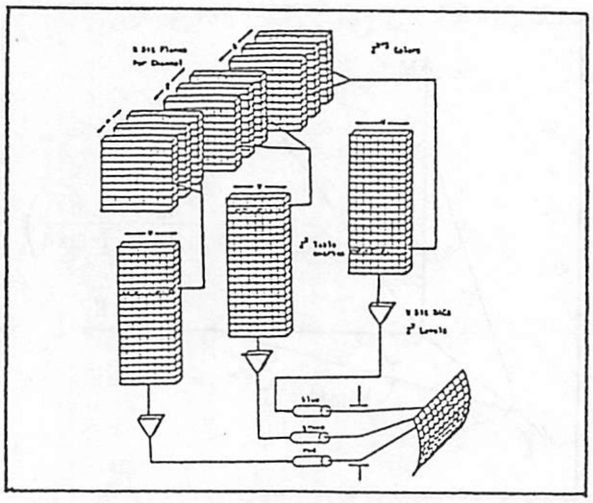


Figure 12 Typical frame buffer functional diagram (from SIGG83b)

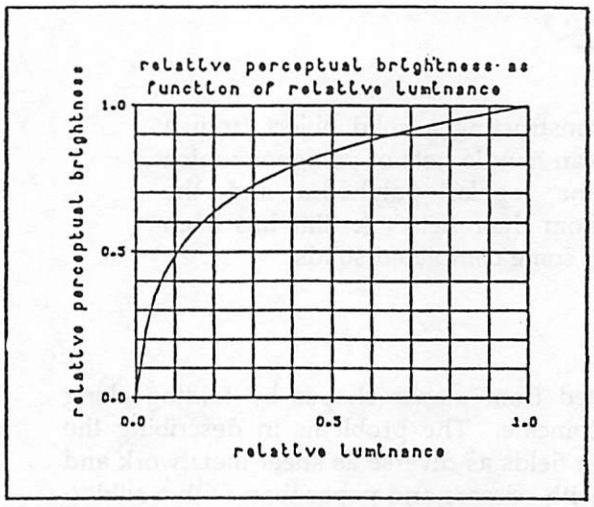


Figure 13 Human visual system response curve

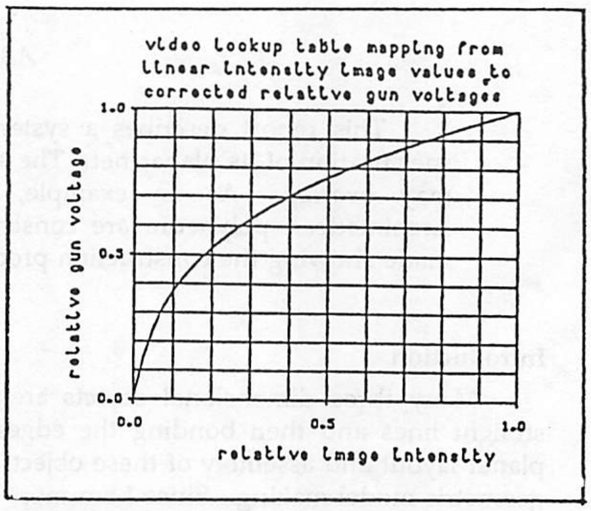


Figure 14 Typical video lookup table for gamma correction of the monitor

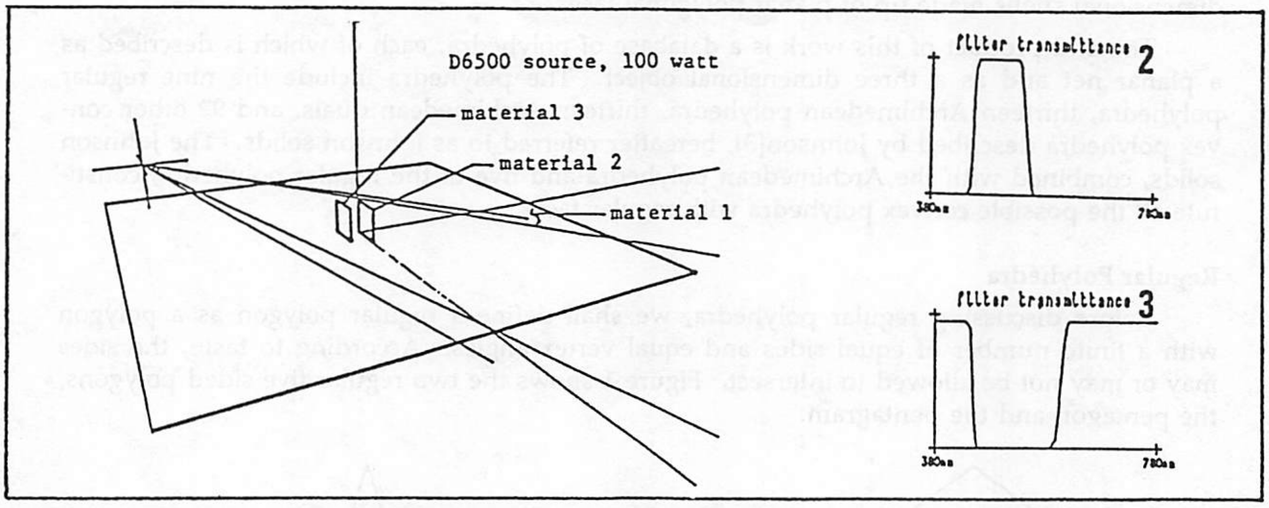


Figure 15 Material curves and test environment for spectral computation experiments (from HALL83a)

Folding Regular Polyhedra

Andrew Hume

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This report describes a system for constructing a solid object from a specification of its planar net. The system can handle nets of polygons which may overlap. As an example, the nine regular polyhedra and the Archimedean polyhedra are constructed from their nets. A film has been made showing the construction process and some completed solids.

Introduction

Many three dimensional objects are fabricated from planar shapes by folding along straight lines and then bonding the edges that coincide. The problems in describing the planar layout and assembly of these objects concern fields as diverse as sheet metalwork and geometric model-making. Since I am more adept with scissors and paper than with a soldering iron, this report will focus on the latter although all the techniques apply equally well to the former. In particular, the focus is on *polyhedra* which can be considered as three-dimensional shells made up of planar polygonal faces.

The end product of this work is a database of polyhedra, each of which is described as a planar net and as a three dimensional object. The polyhedra include the nine regular polyhedra, thirteen Archimedean polyhedra, thirteen Archimedean duals, and 92 other convex polyhedra described by Johnson[3], hereafter referred to as Johnson solids. The Johnson solids, combined with the Archimedean polyhedra and five of the regular polyhedra, constitute all the possible convex polyhedra with regular faces*.

Regular Polyhedra

Before discussing regular polyhedra, we shall define a regular polygon as a polygon with a finite number of equal sides and equal vertex angles. According to taste, the sides may or may not be allowed to intersect. Figure 1 shows the two regular five-sided polygons, the pentagon and the pentagram.

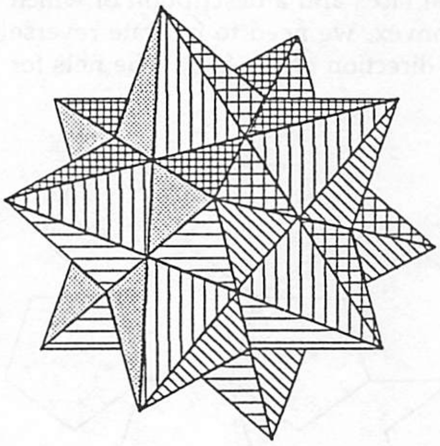
pentagon
pentagram

Figure 1

*excluding the two infinite classes described below (prisms and antiprisms).

We can define a regular polyhedron analogously to a regular polygon; a finite number of congruent regular polygon faces and equal dihedral angles (the angle formed by two faces meeting along an edge) not equal to π radians. A more complete and rigorous treatment can be found in [1].

Nine solids meet this definition: the first five are the Platonic solids, which were known before the time of Euclid, the tetrahedron, cube, octahedron, dodecahedron and icosahedron. The other four are known as the Kepler-Poinsot solids and have intersecting faces. The small and great stellated dodecahedrons were discovered by Kepler (1571-1630) and have pentagram (star) faces. The great dodecahedron and great icosahedron were found by Poinsot (1777-1859) and have pentagonal and triangular faces respectively. The small stellated dodecahedron (shown in Figure 2) and the great dodecahedron are peculiar as they do not satisfy the Euler theorem $F + V = E + 2$ in its normal form.


small stellated dodecahedron

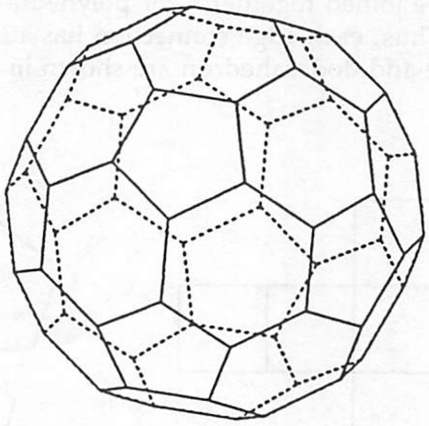

truncated icosahedron

Figure 2

Uniform Polyhedra

Another way to define a regular polyhedron is that its vertex figures are congruent regular polygons. The vertex figure is a polygon formed by joining the midpoints of all the edges meeting at that vertex. Other classes of less regular polyhedra can be generated by relaxing this definition. If the faces are all regular and the vertex figures congruent but not regular, then we get the (facially-regular) Archimedean solids. In this case, the dihedral angles are only equal between congruent pairs of faces. If the vertex figures are regular but not all congruent and all the dihedral angles are equal, then we get the (vertically-regular) Archimedean duals. For historical reasons, the Archimedean duals are largely ignored in discussions of uniform polyhedra.

The Archimedean solids can be divided into three classes. The first class is an infinite set of prisms formed by taking two regular congruent parallel faces and joining corresponding edges with squares. The second class is an infinite set of antiprisms. To construct the antiprism of a given prism, consistently divide each of the rectangular side faces into two triangles by a diagonal and then rotate one of the two regular congruent faces* in the direction such that all the triangular side faces are congruent. The third class is a set of thirteen solids ranging from 8 to 92 faces. One of these, the truncated icosahedron†, is shown in Figure 2.

*through $\frac{\pi}{n}$ where n is the number of sides in the rotated face.

†Yes, it really is a soccer ball! It is also apparently the shape of a new molecule C_{60} (*Nature*, 318, 162-163, 1985).

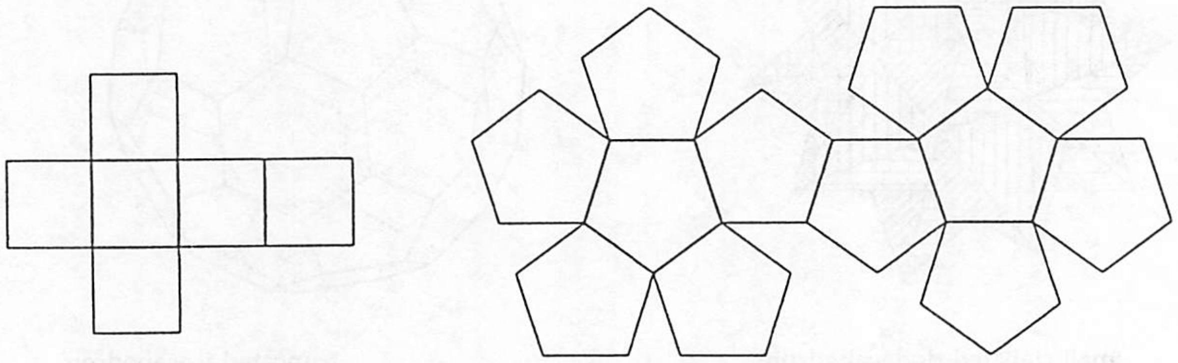
The dual of an Archimedean solid s has its faces associated with the vertices of s. The shape of the face is taken by constructing the dual of the vertex figure at any vertex. The dual polygon is formed by drawing the circle containing all the vertices of the vertex figure and drawing the tangents to this circle at each of the vertices. These tangents are the sides of the dual polygon.

The Database

The database is built in three steps. The first and most tedious step is to describe the planar net of the polyhedron. The second determines the dihedral angle between the faces. The third step combines the information from the first two steps and generates a three dimensional description of the polyhedron.

The Net Database

The net or development of a solid is a set of polygonal faces and a description of which edges are joined together. For polyhedra which are not convex, we need to indicate reverse folds. Thus, each edge connection has a sign showing the direction of the fold. The nets for the cube and dodecahedron are shown in Figure 3.



cube dodecahedron

Figure 3

The net description forms an ASCII file. The descriptions for the above nets are:

```
solid "cube"                           solid "dodecahedron"
0:{4}-45                               0:{5}-180
1:{4} 2:{4} 3{4} 4:{4} 5:{4}           1:{5} 2:{5} 3:{5} 4:{5} 5:{5} 6:{5}
<0.0 2.2> <0.1 3.3> <0.2 4.0>          7:{5} 8:{5} 9:{5} 10:{5} 11:{5}
<0.3 1.1> <5.0 4.2>
                                       <0.0 4.2> <0.1 5.2> <0.2 1.2>
                                       <0.3 2.2> <0.4 3.2> <5.4 6.4>
                                       <11.0 10.2> <11.1 6.2> <11.2 7.2>
                                       <11.3 8.2> <11.4 9.2>
```

Solids are described by a label, a set of face descriptions and a set of edge connections. Faces are shapes (typically regular n-gons denoted by $\{n\}$) followed by an optional rotation in degrees*. Connections are denoted by two edge specifications. An edge is defined by $f.e$ where f is the face and e is the edge on that face. Edge e connects vertices e and $(e+1) \bmod n$ where n is the number of sides of the face. Vertices are numbered counter-clockwise with zero at the north pole (before rotation). Reverse folds are denoted thus $<f.e \,\hat{}\, g.h>$. Faces

*one rotation is useful for orienting the net. Other rotations can be useful as assertions as incorrect rotations are flagged.

which are not regular n-gons are specified by their side lengths and vertex angles and denoted by an alphanumeric identifier.

The Angler

The angler program determines dihedral angles by attempting to fold a planar net into a convex polyhedron. Non-convex polyhedra have to be done by hand. Of course, for the regular and Archimedean solids considered here, there are often simple and elegant constructions for the dihedral angles. However, the folder is designed for the general case. A top level view of the algorithm is:

```
form perimeter
while perimeter not empty
{
        pick a vertex v
        form a solid vertex at v
        update perimeter
}
```

The perimeter is kept as a linked list of edges in counter-clockwise order. Given a vertex, the folder assumes that the two edges on the perimeter which meet at that vertex will coincide. Using the methods described below, it calculates the dihedral angles and rotates the faces (and any other attached faces). The perimeter is then checked and edges that coincide are deleted. Note that this may cause the perimeter to become a number of isolated edge lists.

A vertex is selected according to the heuristic of finding the vertex with the minimum angle between the two free edges. This heuristic may fail as sometimes a missing face will come from another part of the net during folding. For example, the nets for the snub cube and snub dodecahedron have large gaps that are filled by faces from another part of the net. This only becomes apparent after partial assembly. If the angler detects an error or inconsistency, it backtracks and selects another vertex.

Vertex selection for the Johnson solids is done differently. If n faces meet at a vertex, we say that vertex is of order n. If we are determining the dihedral angles between the faces meeting at a vertex of order n, there are at most $n-3$ degrees of freedom. Thus, vertices of order 3 have a unique solution. Because of the symmetries of the Archimedean solids and their duals, all the order 4 and 5 vertices have unique solutions as well. The Johnson solids do not have these symmetries. However, in most cases, by solving the order 3 vertices first, we can reduce the degrees of freedom at higher order vertices to zero and thus solve the entire solid.

In general, the angler checks itself after every vertex it forms and in case of errors, it dumps out the current structure and the perimeter lists of unjoined edges. This catches both bad vertex selection and impossible nets, that is, nets that do not form a solid.

Determining the Dihedral Angle

If all the faces are regular polygons, every vertex is of order 3, 4 or 5; examples are shown in Figure 4. Because the sum of the angles at the vertex must be less than 2π and the smallest angle in any regular polygon is $\pi/3$, the number of faces must be less than 6.

This is not true of the Archimedean duals as their faces are not regular polygons and in fact the hexakis icosahedron (the dual of the great rhombicosahedron) has vertices of order ten. However, as all the dihedral angles for the Archimedean duals are equal, we can use the vertex of mimimum degree. By inspection, the maximum degree vertex is 5 (in the pentakis dodecahedron, the dual of the truncated dodecahedron).

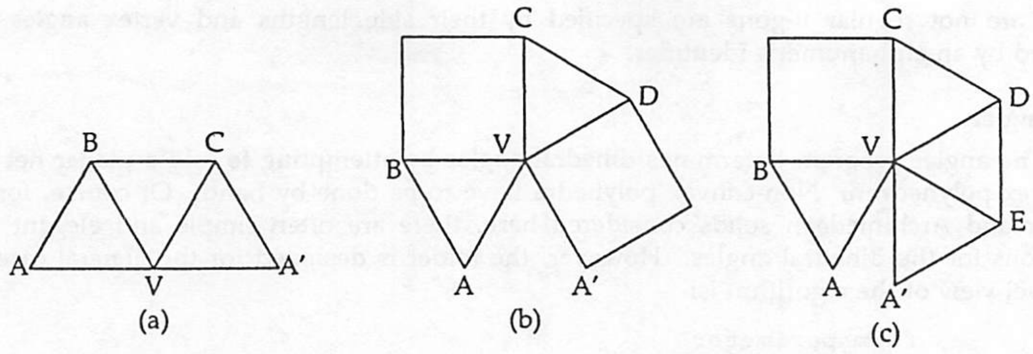


Figure 4

Vertices of order 3 are straightforward to solve. In the example in Figure 4(a), the face VAB is rotated about the edge VB and VA′C about VC until the vertices A and A′ coincide.

Vertices of order 4 form a quadrilateral pyramid with one degree of freedom. As discussed above, for the Archimedean polyhedra and their duals, symmetries reduce the number of possible cases to the following few. In Figure 4(b), the apex will be V and the (rectangular) base ABCD. The base will always be one of four shapes. These shapes are shown in Figure 5(a) and 5(b). Assuming all the edges are of length 1, the parameter z is one of two values, $\sqrt{2}$ or ϕ, where ϕ is the golden ratio ($\approx$1.618). The two dihedral angles can be found easily by spherical trigonometry.

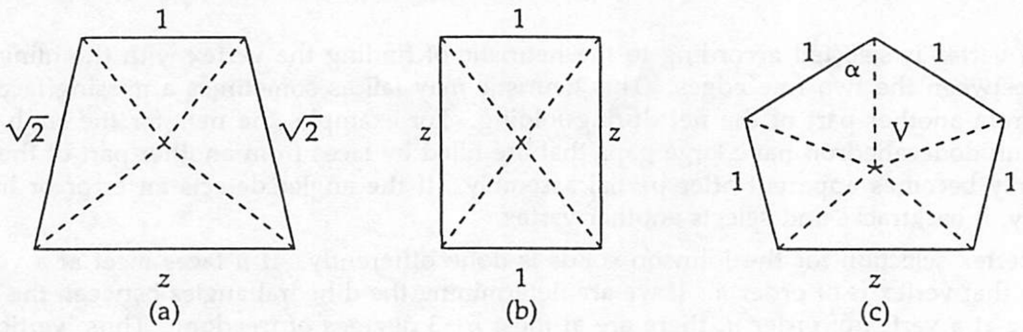


Figure 5

Vertices of order 5 are treated analogously to vertices of order 4. In the example of Figure 4(c), the pyramid has apex V and base ABCDE. The shape of the base of the pyramid is shown in Figure 5(c). The half-angle α can be found by solving the cubic $\sin\alpha - \sin 3\alpha = \frac{z}{2}$.

The Folder

The folder program treats the polyhedron as a set of hinged shells. Initially, every face is a shell. At every hinge, one of the two shells is rotated through the specified angle. Checks are made for coinciding edges and all shells with coincident edges are merged into one shell. At the end, there should be only one shell.

Current Status

The database has complete entries for the regular, Archimedean and Archimedean dual polyhedra. Nets have been entered for all of the Johnson solids and some of these have dihedral angles and have been folded. The numerical error in the coordinates (stored on a VAX as double precision) is smaller than 10^{-14}.

Future work goes in two directions: determining an algorithmic way to find the dihedral angles for awkward Johnson polyhedra and presenting the data for each polyhedron symbolically. For example, specifying the dihedral angle for the dodecahedron as $\pi - \tan^{-1} 2$ rather than the standard 116° 34′.

Conclusions

It is possible to construct a variety of solids using only their planar nets and with no knowledge about the final structure other than convexity. In addition, many solids (such as the regular stellated polyhedra) with concave vertices can be handled by means of reverse folds.

Indeed, constructing the solids by folding rather than by analytic methods has some advantages, as the mapping between faces in the planar and solid representations is directly derived. For example, given a geographical database, it is possible to project a world map onto any of the convex regular polyhedra. Each face is tangential to the inter-sphere and using a gnomonic projection, we can obtain nets that fold into polyhedral world globes. Such a net for the truncated icosahedron is shown in Figure 6.

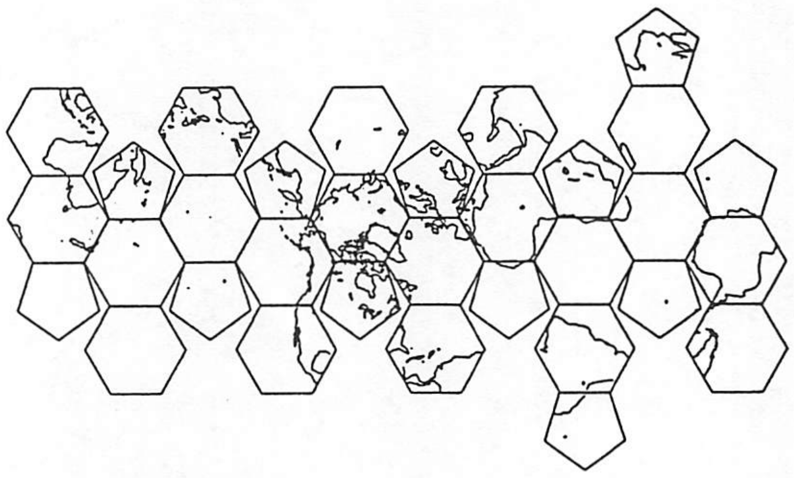


Figure 6

Acknowledgements

My interest in polyhedra and models stems directly from a fine book on geometric models by Cundy and Rollett[2]. Doug McIlroy provided encouragement and the map software. Tom Duff wrote the polygon renderer used in making the film. Peter Weinberger provided his inspiring visage.

References

[1] Coxeter, H.S.M., *Regular Polytopes* (2nd Ed), Macmillan, New York, 1963.

[2] Cundy, H.M. and Rollett, A.P., *Mathematical Models* (2nd Ed), Oxford University Press, Oxford, 1961.

[3] Johnson, N.W., Convex Polyhedra with Regular Faces, *Canadian Journal of Mathematics*, XVIII, 1966.

List of Attendees

Rick Adams
Center for Seismic Studies
1300 North 17th Street
Suite 1450
Arlington, VA USA
703-276-7900
rick@seismo.css.gov

Cecilia Aragon
Digital Equipment Corp.
100 Hamilton
Palo Alto, CA USA
415-853-6746
aragon@decwrl

Ken Arnold
University of California, San Francisco
9265 Medical Science Bldg.
San Francisco, CA USA
415-476-5128
ucbvax!arnold

James Batson
Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA USA
408-973-3467

William Beblo
Bell Communications Research
290 W. Mt. Pleasant Ave.
Room 1D-148
Livingston, NJ USA
201-740-4421
ulysses!gamma!wb

Richard Becker
AT&T Bell Laboratories
600 Mountain Ave.
Room 2C 259
Murray Hill, NJ USA
201-582-5512
research!rab

Jon Bentley
AT&T Bell Laboratories
Room 2C-317
Murray Hill, NJ USA
201-582-2315
research!jlb

Charles Bigelow
Bigelow & Holmes
15 Vandewater Street
San Francisco, CA USA
415-788-8973
cab@su-ai.arpa

Dennis Biringer
AVCO Research Lab.
P.O. Box 261
Puunene, Maui, HI USA
808-871-6212
biringer@anl-mcs

Reidar Bornholdt
Columbia University
7-444 P & S
630 W. 168th St.
New York, NY USA
212-305-7411
usenix!reidar

Charles Brauer
Fairchild Research Center
4001 Miranda
MS: 30-300
Palo Alto, CA USA
415-858-4572

David Brown
Digital Equipment Corp.
100 Hamilton Avenue
Palo Alto, CA USA
415-853-6712
decwrl!djb

List of Attendees

Victoria Brown
Genetech, Inc.
P.O. Box 1488
Pacifica, CA USA
415-994-6862
ucbvax!ucsfcgl!genie!clinical!vlb

Eric Brunner
737 56th Street
Oakland, CA USA
415-654-6002
brunner@ucbisis

Greg Chesson
Silicon Graphics, Inc.
2011 Stierlin Road
Mountain View, CA USA
415-960-1980

Arvin Conrad, Jr.
Menil Foundation
1519 Branard Street
Houston, TX USA
713-528-1345

Mark Cutter
Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA USA
408-973-3467

Judith DesHarnais
USENIX Conference Office
P.O. Box 385
Sunset Beach, CA USA
213-592-3243

Marc Donner
IBM Research
P.O. Box 218
Yorktown Heights, NY USA
donner.yktvmx.ibm@csnet-relay.arpa

Tom Duff
AT&T Bell Laboratories
600 Mountain Avenue
2C-425
Murray Hill, NJ USA
201-582-6485
research!td

Robert Ellis
G.E./Calma Company
525 Sycamore Dr.
Milpitas, CA USA
408-434-4925
ucbvax!calma!ellis

Melvin Ferentz
Rockefeller University
1230 York Avenue
New York, NY USA
212-570-8925
mel@rockefeller.arpa

Rob Fish
Bell Communications Research
2E-237, 445 South Street
Morristown, NJ USA
bellcore!joevax!robf

Allan Frankel
Megatech
9645 Scranton Road
San Diego, CA USA
619-455-5590

Adrian Freed
IRCAM
22 Sirard Lane
San Rafael, CA USA
415-485-9790
ucbvax!dagobah!adrian

Ann Geoffrion
SRI International
333 Ravenswood Avenue
Menlo Park, CA USA
geoffrion@SRI-stripe.arpanet

Stephen Glaser
Prime Computer
492 Old Connecticut Path
Framingham, MA USA
617-626-1700
harvard!prime!steveg

List of Attendees

Steven Glassman
Acorn Research Center
5 Palo Alto Square
Suite 910
Palo Alto, CA USA
415-424-1114
decwrl!acorn!glassman

James Gosling
Sun Microsystems
2250 Garcia Avenue
MS 5-40
Mountain View, CA USA
415-960-7242
ucbvax!sun!jdg

Charles Grant
Lawrence Livermore National Laboratory
P.O. Box 2551
Livermore, CA USA
415-422-7278

Satish Gupta
IBM
P.O. Box 218
Yorktown Heights, NY USA
914-945-2721
sgupta.yktvmx@ibm

Paul Haeberli
Silicon Graphics, Inc.
2011 Stierlin Road
Mountain View, CA USA
415-960-1980
ucbvax!olympus!paul

Roy Hall
Graphic Software Services
5270 Bellingham Ave.
N. Hollywood, CA USA
213-650-2434
randvax!hollywood!roy

Pat Hanrahan
DEC Systems Research Center
130 Lytton Avenue
Palo Alto, CA USA

Marianne Hsiung
Apple Computer, Inc.
20525 Mariani Avenue
MS 22G
Cupertino, CA USA
408-973-3550

Conrad Huang
University of California, San Francisco
9265 Medical Science Bldg.
School of Pharmacy
San Francisco, CA USA
415-476-5128
conrad@ucsf-cgl.arpa

Andrew Hume
Bell Laboratories
Room 2C 423
Murray Hill, NJ USA
201-582-6262
research!andrew

Lou Katz
Metron Computerware, Ltd.
3317 Brunell Drive
Oakland, CA USA
415-530-8870
ucbvax!lou

Ken Knowlton
Networked Picture Systems, Inc.
3255 Scott Blvd.
Bldg. 7, Suite H
Santa Clara, CA USA
408-748-1677

Jeff Langer
AT&T Bell Laboratories
600 Mountain Ave., 3C-418
Murray Hill, NJ USA
201-582-2455
allegra!jeff

Peter Langston
Bellcore
435 South Street
2C 416
Morristown, NJ USA
201-582-7248
bellcore!psl

List of Attendees

Dave Levine
Lucasfilm, Ltd.
P.O. Box 2009
San Rafael, CA USA
415-485-5019
usenix!ucbvax!dagobah!dl

Creon Levit
NASA Ames Research Center
MS 233-1
Moffett Field, CA USA
415-694-6410
creon@ames-nas

Gregory MacNicol
Digital Design
222 S. Branciforte
Santa Cruz, CA USA
408-426-0403

Arthur Malan
Tektronix, Inc.
P.O. Box 500
MS 47-660
Beaverton, OR USA
503-627-4184

Alan Marr
Sun Microsystems
MS 5-40
2500 Garcia Avenue
Mountain View, CA USA
415-960-7261
amarr@sun

David Meyers
University of California, Santa Cruz
P.O. Box 8444
Santa Cruz, CA USA
408-427-0963

Fred Mikami
California Computing Resources
20941 Devonshire Street
Chatsworth, CA USA
818-709-2681

Gregory Millar
Visual Engineering, Inc.
2680 No. First St., Suite 200
San Jose, CA USA
408-945-9055

Eugene Miya
NASA Ames Research Center
MS 241-3
Moffett Field, CA USA
415-694-6453

Richard Morin
Canta Forda Computer Laboratory
P.O. Box 1488
Pacifica, CA USA
415-994-6860
ucbvax!ucsfcgl!genie!cfcl!rdm

Richard Mossman
Pacific Bell
120 Montgomery Street
Room 570
San Francisco, CA USA
415-774-8836

Lars Nyland
Microelectronics Center of NC
P.O. Box 12889
Research Triangle Park, NC USA
919-248-1960
lsn@mcnc

William Ortel
AT&T Information Systems
307 Middletown-Lincroft Rd.
Room 3L-321
Lincroft, NJ USA
201-576-3242

Andy Pfiffer
Cornell University
Cornell Theory Center
265 Olin Hall
Ithaca, NY USA
607-256-8686
andy@devvax.tn.cornell.edu

List of Attendees

Floyd Phillips
Calculon Corp.
1301 Piccard Dr.
Rockville, MD USA
301-258-5450

David Plaisted
American Multi-Tech Corp.
2 Lakeview Road
Lake Nelson
Piscataway, NJ USA
201-463-8566

Michael Podhorodecki
Labtam International Pty., Ltd.
43 Malcolm Road
Braeside, Vic. Australia
+61 3587 1444

Michael Potel
Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA USA
408-973-3467

Gail Rein
Microelectronics &
Computer Technology Corp.
9430 Research Blvd.
Austin, TX USA
512-834-3303
rein@mcc

Craig Rodine
AT&T Bell Laboratories
600 Mountain Avenue
Room 2C-421
Murray Hill, NJ USA
201-582-7248
research!crr

David Rosenthal
Sun Microsystems
2550 Garcia Avenue
Mountain View, CA USA
415-960-7569
sun!devnull!dshr

Peter Ross
Intel Scientific Computer
15201 N.W. Greenbriar
Beaverton, OR USA
503-629-7679

Patrick Ryan
McMaster University
Dept. of Computer Science
Hamilton, Ont. Canada
416-525-9140

Sunil Saxena
Counter-Point Computers
2127 Ringwood Avenue
San Jose, CA USA
408-434-0190
oblio

Carlo Sequin
University of California, Berkeley
Dept. of EECS
Computer Science Division
Berkeley, CA USA
415-642-5103
sequin@berkeley

Alok Singhania
Digital Research, Inc.
P.O. Box DRI
Monterey, CA USA
408-646-6304

Michael Sleator
2281 NW Hoyt
Portland, OR USA
503-224-0196
tecktronix!psu-cs!darkzone!sleat

Irwin Sobel
Hewlett Packard
3500 Deer Creek Road
Palo Alto, CA USA
415-857-5774

List of Attendees

Cheryl Stewart
Cornell University
Cornell Theory Center
265 Olin Hall
Ithaca, NY USA
607-256-8686
cheryl@devvax.tn.cornell.edu

Ray Swartz
Berkeley Decision Systems
150 Belvedere Terrace
Santa Cruz, CA USA
408-458-0500

Spencer Thomas
University of Utah
Computer Science Dept.
Salt Lake City, UT USA
801-581-3095
decvax!utah-cs!thomas

Rebecca Thomas
UNIX World Magazine
241 Cascade Drive
Fairfax, CA USA
415-258-0957

David Tristram
NASA Ames Research Center
MS 233-1
Moffett Field, CA USA
415-694-6410
dat@ames-nas

Margaret Twomey
NYIT Computer Graphics Lab
Wheatley Road
Old Westbury, NY USA
516-686-7644

Vincent Vaillancourt
AT&T Bell Laboratories
Room 1B-128
260 Cherry Hill Road
Parsippany, NJ USA
201-299-3224
ihnp4!vilya!vrv

Turner Whitted
University of North Carolina
New West Hall 035a
Chapel Hill, NC USA
919-962-0195
jtw@unc

David Yost
Laurel Arts
8464 Kirkwood Drive
Los Angeles, CA USA
213-650-1089
randvax!hollywood!day

James Zamiska
FMC Corporation
1185 Coleman Avenue
Box 580
Santa Clara, CA USA
408-289-3139

USENIX Association
P.O. Box 7
El Cerrito, CA 94530

First Class Mail

FIRST CLASS MAIL
U.S. POSTAGE PAID
El Cerrito, CA 94530
Permit No. 87